准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->JSCORE->day04

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->JSCORE->day04

如果因为某些原因错过直播讲解，不用着急！直接看小程序中视频的回看即可。

如果tmooc不能用了，立刻看小程序视频，小程序视频比tmooc要全，甚至有扩展的新知识。

如果学有余力，将来有能力拿更高薪资，小程序中所有视频必须都要看。但是不必现在都看完。只要在笔试面试前都看完即可！

问题: 个人版小程序有功能限制：禁止在线播放视频

解决: 长按视频连接地址，完整复制视频连接地址和验证码到浏览器打开。通过百度网盘下载后观看。

3. 今日课程中包含的英语单词的翻译：小程序->单词->JSCORE->day04

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例，至少写三遍:

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩18天）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

谁问的问题越多！谁的培训费交的越值！

正课: OOP

1. 什么是面向对象:

2. 封装:

3. 继承:

4. 多态:

一. 什么是面向对象:

1. 什么是: 程序中都是先用对象结构来集中描述现实中一种事物的属性和功能。然后再按需访问对象中保存的属性和功能。

2. 为什么: 便于大量数据的管理和维护

3. 何时: 今后所有程序都要使用面向对象的思想开发

4. 如何: 三步/面向对象三特点: 封装，继承，多态

二. 封装:

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

1. 什么是: 将一个事物的属性和功能集中定义在一个对象内

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思想开发，第一步都是要先封装对象。然后才是按需使用不同对象中的属性和功能。

4. 如何创建一个对象:

(1). var 对象名={

属性名: 属性值,

... : ... ,

方法名: function(){

... ...

}

}

(2). 一个事物的属性值，会成为程序中对象里的一个属性。事物的一个功能(会做的一件事)，就会成为对象中的一个方法(function(){ ... })。——{}中的一切属性和方法统称为这个对象的成员.

5. 如何访问对象中的成员:

(1). ~~错误做法: 不能直接将对象中的属性当做普通全局变量访问:~~

~~console.log(属性名) //报错~~

(2). 正确做法: 必须先用对象名变量找到对象，再用.操作符，进入对象内访问对象的成员：

a. 访问对象中的属性: 对象名.属性名

b. 访问对象中的方法: 对象名.方法名()

6. this:

(1). 问题: 对象自己的方法中，想使用对象自己的属性，竟然报错，说属性名未定义！

(2). 原因: 因为对象不是作用域，所以对象中的方法的作用域链里不包含当前对象本身。所以，方法中不加任何.前缀的变量只能在方法内和window中查找。无法自动进入当前对象中查找。所以，即使对象自己的方法，想访问对象自己的属性，也不能直接使用！

(3). 解决:

a. 不好的解决方法: 方法中如果想用当前对象的属性，可加上"对象名.属性名"

b. 问题: 方法内写死外部对象的对象名——紧耦合。一旦将来外部的对象名发生变化，我们被迫要记得手动修改内部方法中写死的对象名——极其不便于维护！——应该极力避免！

c. 好的解决方法: 今后只要对象自己的方法想使用对象自己的属性，必须写成"this.属性名"。

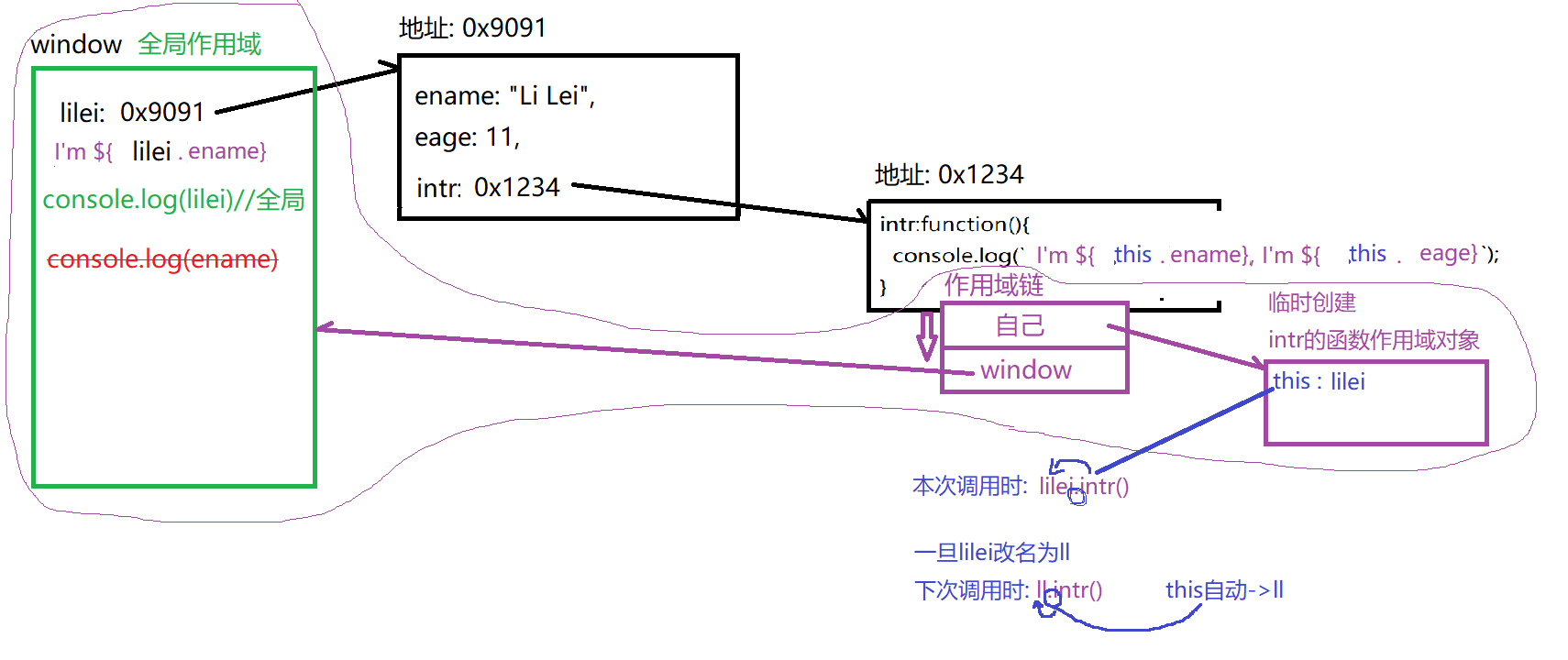
1). 什么是this: this是专门自动引用正在调用当前函数的.前的对象 的关键词

2). this关键词保存在函数临时创建的函数作用域中

3). 每次调用函数时，this自动先找"."，然后再自动指向.前的对象

比如: lilei.intr() this->.前的lilei， ll.intr() this->.前的ll

4). 好处: 方法内即使不写死对象名，也可以通过this自动获得.前当前对象现在的名字。执行时，和写死对象名效果完全一样！



(4). 总结: 今后只要对象自己的方法想使用对象自己的属性，必须加"this.属性名"。

7. 示例: 定义对象描述一个学生lilei的属性和会做的事儿:

|  |
| --- |
| <script>  //创建一个学生对象，来描述现实中一个学生的属性值和会做的事儿  //学生姓名: Li Lei  //学生年龄: 11  //学生会自我介绍: 李磊会说: I'm Li Lei, I'm 11  var lilei={ //对象的{}不是作用域  //属性名 : 属性值  ename: "Li Lei",  eage: 11,  //功能/方法: 因为功能是指会做一件事，所以功能被定义为一个函数  intr:function(){  console.log(`I'm ${this.ename}, I'm ${this.eage}`); //报错: ename未定义  }  }  //想输出李磊的年龄：  console.log(`李磊的年龄: ${lilei.eage}`);  //想请李磊做自我介绍:  lilei.intr();  //过了一年，李磊长了一岁  lilei.eage++;  //再输出李磊的年龄  console.log(`李磊的年龄: ${lilei.eage}`);  //再请李磊做自我介绍  lilei.intr();  </script>  运行效果：  李磊的年龄: 11  I'm Li Lei, I'm 11  李磊的年龄: 12  I'm Li Lei, I'm 12 |

8. 其实创建对象可以有: 3种方式:

(1). 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

}

}

(2). 用new Object()创建一个对象:

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. 如何: 2步

1). 先创建一个空对象

var 对象名=new Object()

2). 给空对象中强行添加新属性和方法

对象名.属性名=属性值;

对象名.方法名=function(){ ... }

b. 揭示了一个本质:

1). {}其实就是new Object()的简写

2). js中对象底层也是关联数组

i. 都保存的是名值对儿的组合

ii. 都可用["下标名/属性名"]访问自己的成员

也都可以简写为"对象.属性名"方式

大部分情况不用在傻傻的区分该用.还是该用[""]，因为.就是[""]

特例: 如果属性名不是写死的，而是来自于一个变量，就不能用.，只能用[变量名]，且还不能加""

iii. 都可以随时往不存在的位置添加新成员：通过强行赋值的方式！

所以将来要想给对象或关联数组添加新成员，唯一的方式只有：强行赋值

iv. 强行访问对象或数组中一个不存在的位置，不会报错！会返回undefined

所以将来要判断一个对象或关联数组中是否包含某个成员：

对象.成员名!==undefined 返回true，说明包含该成员

返回false，说明不包含该成员！

v. 遍历对象或关联数组中的成员，都可用: for in循环

回顾: for(var 变量名 in 对象或关联数组){

in会自动依次获得对象或关联数组中每个属性的属性名，保存到in前的变量中

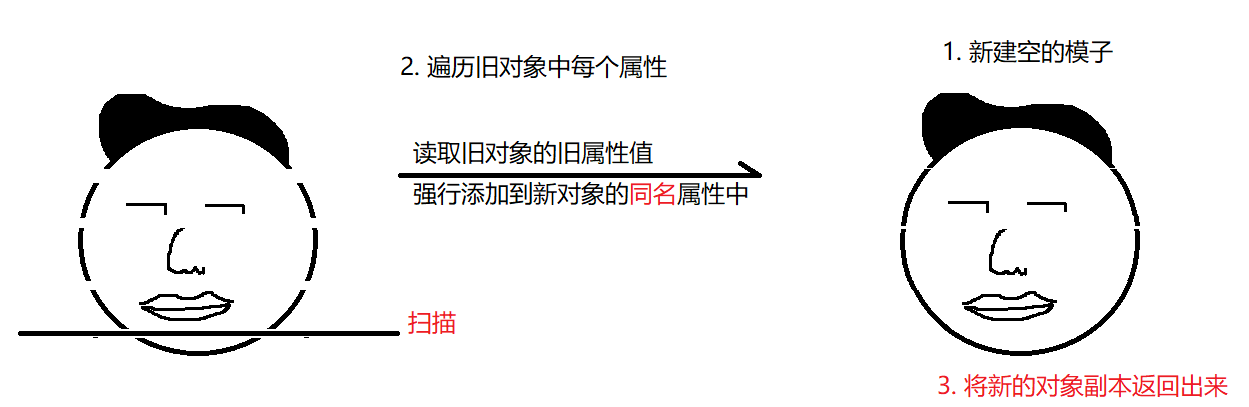
然后，我们就可以利用"对象[变量名]"读取出对象或关联数组中每个属性的属性值。

}

c. 示例: 比较对象和关联数组

|  |
| --- |
| <script>  var lilei={ //一个对象lilei  ename:"Li Lei",  eage:11  };  var ym=[]; //一个关联数组ym  ym["ename"]="杨幂";  ym["eage"]=30;  //输出对象和关联数组在内存中的存储结构，比较相似点  console.log(lilei);  console.log(ym);  //想访问lilei的ename属性  console.log(lilei["ename"], lilei.ename)  //想访问ym的ename属性  console.log(ym["ename"], ym.ename)  //想给李磊和ym都添加一个新属性:班级className  lilei["className"]="初一2班";  ym["className"]="表演系2班";  //再次输出两个对象，观察新属性是否强行添加成功  console.log(lilei);  console.log(ym);  //想获取李磊和ym的className属性  console.log(lilei.className, ym.className);  //想获得李磊和ym的不存在的salary属性  console.log(lilei.salary, ym.salary);  //分别遍历lilei和ym中都有哪些属性，属性值分别是什么:  for(var a in lilei){  console.log(`属性名:${a}`)  console.log(`属性值: ${lilei[a]}`)  //console.log(`属性值:${lilei.a}`) //错误  //a是一个循环变量，每一轮的值都不一样，都在不停变化  //lilei.a会被翻译为lilei["a"]，如果变量a被包在""中，就不再是变量了。而是一个普通的写死的属性名"a"了！也就无法变化了。  //如果lilei[a]，每一轮循环中，lilei[a]，分别会变成lilei["ename"]、lilei["eage"]、lilei["className"]，因为不加""的a是变量，可以不断变化。  }  for(var b in ym){  console.log(`属性名:${b}`);  console.log(`属性值: ${ym[b]}`)  //console.log(`属性值:${ym.b}`) //错误  //b也是一个循环变量，每一轮的值都不一样，都在不停变化  //ym.b会被翻译为ym["b"]，如果变量b被""包裹，就不再是变量了。而是一个普通的写死的属性名"b"，也就无法发生变化了  //如果ym[b]，每一轮循环， ym[b]，分别会变为ym["ename"]、ym["eage"]、ym["className"]。因为不加""的b是变量，可以不断变化。  }  </script>  运行效果：  {ename: "Li Lei", eage: 11}  [ename: "杨幂", eage: 30]  Li Lei Li Lei  杨幂 杨幂  {ename: "Li Lei", eage: 11, className: "初一2班"}  [ename: "杨幂", eage: 30, className: "表演系2班"]  初一2班 表演系2班  undefined undefined  属性名:ename  属性值: Li Lei  属性名:eage  属性值: 11  属性名:className  属性值: 初一2班  属性名:ename  属性值: 杨幂  属性名:eage  属性值: 30  属性名:className  属性值: 表演系2班 |

d. 示例: 克隆一个对象:



|  |
| --- |
| <script>  var lilei={  sname:"Li Lei",  sage:11  };  //定义一个函数clone()  //接收一个原对象  //经过加工和处理  //返回一个一模一样的新对象  function clone(oldObj){  //1. 先创建一个空对象等着！  var newObj={};  //2. 遍历旧对象中每个属性  for(var key in oldObj){  //每遍历旧对象中一个属性，都要为新对象强行一个同性的新属性，属性值等于旧对象中当前同名属性值  newObj[key]=oldObj[key];  }  //3. 返回克隆好的新对象  return newObj;  }  //测试:  var lilei2=clone(lilei);  console.log(lilei);//长的一模一样  console.log(lilei2);//长的一模一样  console.log(lilei==lilei2); //false 说明两个对象地址不同！是两个独立的存储空间  var hmm={  sname:"Han Meimei",  sage:12  }  var hmm2=clone(hmm);  console.log(hmm);//长的一模一样  console.log(hmm2);//长的一模一样  console.log(hmm==hmm2); //false  </script>  输出结果:  {sname: "Li Lei", sage: 11}  {sname: "Li Lei", sage: 11}  false  {sname: "Han Meimei", sage: 12}  {sname: "Han Meimei", sage: 12}  false |

e. 问题: 前两种创建对象的方式，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，代码很繁琐的！

(3). 使用构造函数反复创建多个相同结构的对象:

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

a. 什么是构造函数: 描述同一类型所有对象的统一结构的函数

b. 为什么: 为了重用对象结构

c. 何时: 只要反复创建多个相同结构的对象时，都要先定义构造函数，来描述一类对象的统一结构。

d. 如何: 2步:

1). 定义构造函数:

function 类型名(形参变量列表){

this.属性名=形参变量1;

this.属性名=形参变量2;

~~this.方法名=function(){ ... }~~

}

2). 用new调用构造函数反复创建同一类型的多个相同结构的对象

var 对象名=new 类型名(实参值列表)

e. new做了4件事（构造函数的原理）

1). new先创建一个新的空对象

2). 设置新对象"\_ \_proto\_ \_"属性自动继承构造函数的原型对象

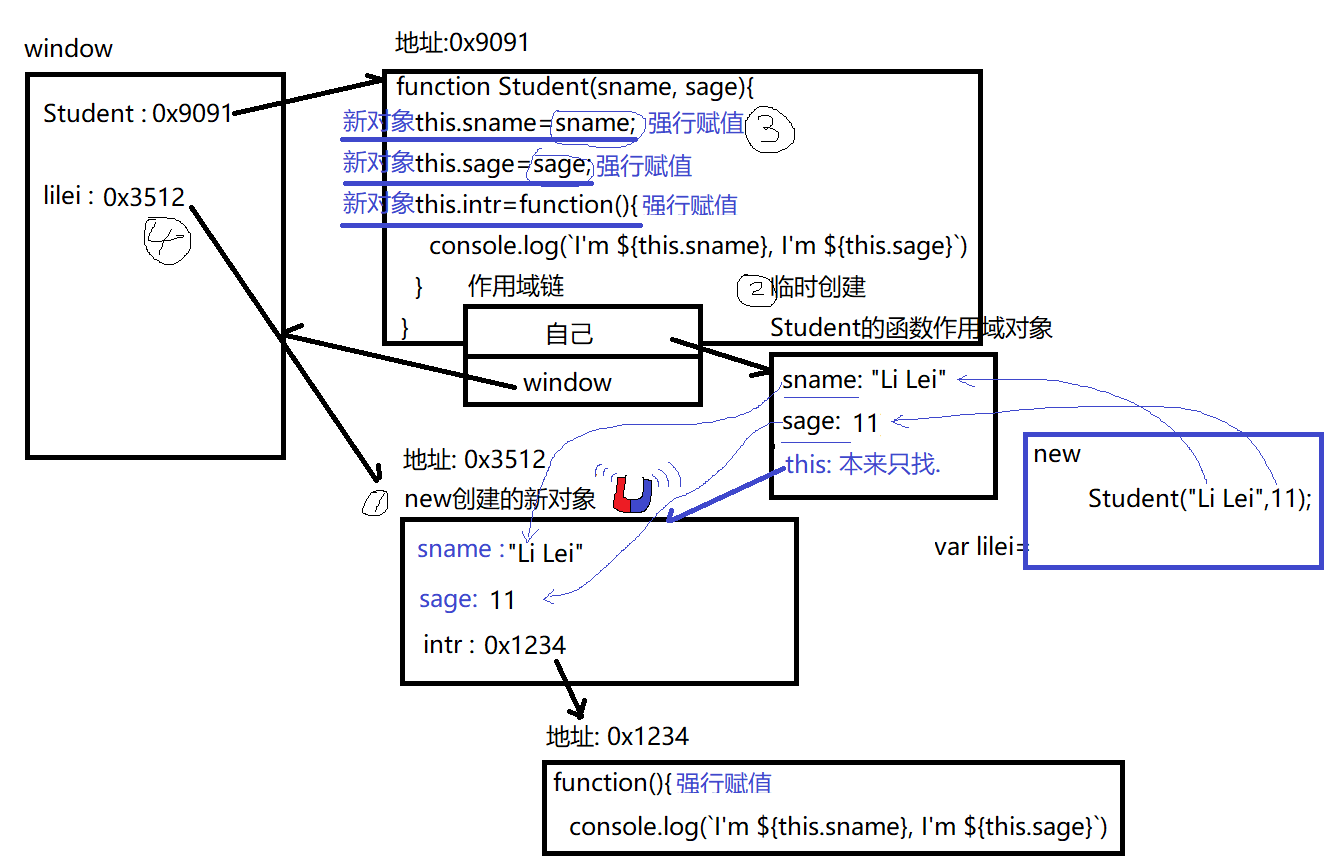
3). 用new调用构造函数:

i. 将实参值传给构造函数的形参变量

ii. 自动将构造函数中所有this都吸引到new创建的新对象来

结果，每一句this.属性名=形参都成了强行给新对象赋值新属性

4). new会返回新创建的对象地址保存给=左边的变量



f. 示例: 创建构造函数Student()描述所有学生对象统一结构，并创建两个具体的学生对象lilei和hmm

|  |
| --- |
| <script>  //想创建两个学生:  //lilei: sname: Lilei, sage: 11, 会自我介绍: I'm Li Lei, I'm 11  //hmm: sname: Han Meimei, sage: 12, 会自我介绍: I'm Han Meimei, I'm 12  //总结: 所有学生都有sname和sage两个属性，所有学生都有intr()方法做自我介绍  //所以:  function Student(sname, sage){  this.sname=sname;  this.sage=sage;  this.intr=function(){  console.log(`I'm ${this.sname}, I'm ${this.sage}`)  }  }  //用new调用Student()构造函数，创建lilei对象，并将李磊的属性值保存进将要创建的lilei对象中  var lilei=new Student("Li Lei",11);  //用new调用Student()构造函数，创建hmm对象，并将韩梅梅的属性值保存进将要创建的hmm对象中  var hmm=new Student("Han meimei",12);  console.log(lilei);  console.log(hmm);  </script>  运行结果:  Student {sname: "Li Lei", sage: 11, intr: ƒ}  Student {sname: "Han meimei", sage: 12, intr: ƒ} |

g. 为什么构造函数中所有规定的属性和方法前都要有this.?

为了让this和将来的new配合，将构造函数中所有规定的属性和方法通过强行赋值的方式，添加到将来new创建的新对象中。

h. 构造函数的优点: 重用代码！

i. 构造函数有问题: 浪费内存——如果把方法定义包含在构造函数中。那么每调用一次构造函数，都会重复创建一次方法函数对象的副本。

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. 什么是: 父对象的成员子对象无需重复创建就可直接使用！

2. 为什么: 既可重用代码，又可节约内存！

3. 何时: 今后只要同一类型多个子对象都需要同一个方法时，这个方法不要定义在构造函数中，应该定义在所有孩子的父对象中一份即可！所有孩子，就可通过继承关系，直接使用！

4. 如何:

(1). 从此构造函数中不能包含任何方法的定义，只包含属性的定义。

(2). 所有子对象都需要的方法，应该集中保存在这个类型的原型对象中

5. 原型对象:

(1). 什么是: 专门为同一类型下所有子对象集中保存共有方法的父对象

(2). 如何创建: 原型对象不是我们自己手工创建的，而是买一增一：

当我们定义构造函数时，其实悄悄的赠送了我们一个空的原型对象

(3). 何时/如何建立继承关系: new的第2步:

a. new为每个子对象自动添加了一个"\_ \_proto\_ \_"，指向当前子对象的父对象

b. new将子对象的"\_ \_proto\_ \_"，自动指向构造函数的原型对象

(4). 结果: 从此，凡是被new出的子对象，都默认继承自构造函数的原型对象。凡是保存在构造函数的原型对象中的共有方法，所有子对象可无条件无需创建就可直接使用。

(5). 当子对象调用一个自己没有，但是却保存在原型对象中的共有方法时，程序首先在子对象自己身上找。如果在自己身上找不到，就会自动延"\_ \_proto\_ \_"关系去父对象查找！

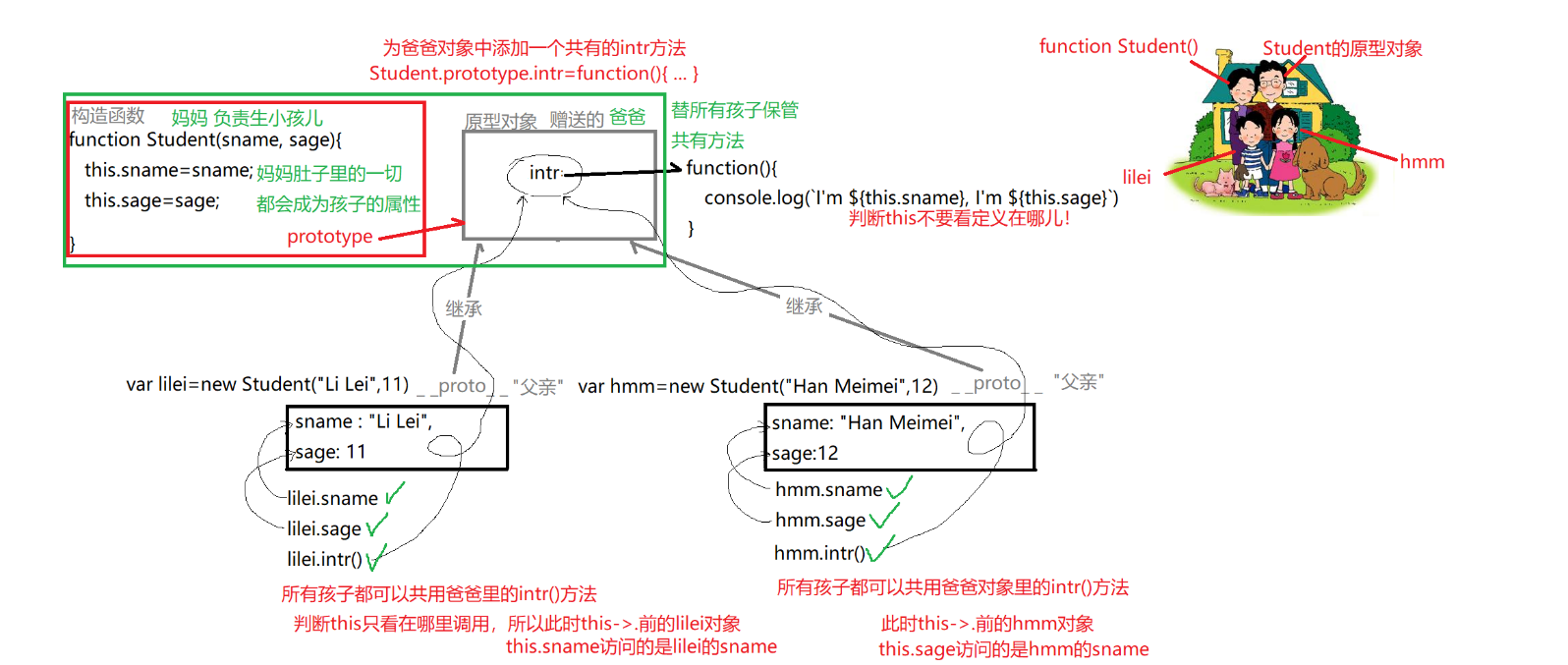
(6). 总结: 今后所有子对象都需要的共有方法应该集中添加到构造函数的原型对象中保存。

(7). 如何将共有方法添加到构造函数的原型对象中: 唯一正确且稳妥的办法:

构造函数都有一个prototype属性，默认指向赠送的原型对象

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

原型对象 强行赋值



6. 示例: 为Student的原型对象添加共有方法，并让子对象调用共有方法

|  |
| --- |
| <script>  //先定义构造函数妈妈，但是不要包含方法定义！  function Student(sname, sage){  this.sname=sname;  this.sage=sage;  }  //输出附赠的空的原型对象  console.log(Student.prototype);//看到一个空对象  //向原型对象中添加intr方法: 强行赋值  Student.prototype.intr=function(){  console.log(`I'm ${this.sname}, I'm ${this.sage}`)  }  //妈妈生了两个孩子  var lilei=new Student("Li Lei",11);  var hmm=new Student("Han meimei",12);  console.log(lilei);  console.log(hmm);  // 李磊 的 爹 是不是 Student的原型对象  console.log(lilei.\_\_proto\_\_==Student.prototype); //true  // 韩梅梅 的 爹 是不是 Student的原型对象  console.log(hmm.\_\_proto\_\_==Student.prototype); //true  lilei.intr();  hmm.intr();  </script>  运行结果：  {constructor: ƒ}  Student {  sname: "Li Lei",  sage: 11,  \_\_proto\_\_:  intr: ƒ ()  }  Student {  sname: "Han meimei",  sage: 12,  \_\_proto\_\_:  intr: ƒ ()  }  true  true  I'm Li Lei, I'm 11  I'm Han meimei, I'm 12 |

总结: （不用背！记不住也没关系！只要保证用的时候可以找到就行了！）

1. String正则相关的函数: 3件事

(1). 查找关键词: 4种情况

a. 查找一个固定的关键词出现的位置:

var i=str.indexOf("关键词", fromi)

b. 用正则模糊查找多种敏感词的位置:

var i=str.search(/正则/i)

c. 查找关键词的内容:

1). 如果只查找一个关键词的内容和位置：

var arr=str.match(/正则/i)

2). 如果查找所有关键词的内容，不关心位置:

var arr=str.match(/正则/ig)

d. 既查找每个关键词的内容，又查找每个关键词的位置:

reg.exec()

(2). 替换关键词: 2种情况:

a. 简单替换: str=str.replace(/正则/ig, "新值")

b. 高级替换: str=str.replace(/正则/ig, function(keyword){

return 根据keyword不同，动态选择不同的新值返回并替换到字符串

})

c. 删除关键词: str=str.replace(/正则/ig, "")

(3). 切割字符串: 2种情况:

a. 简单切割: var arr=str.split("切割符")

b. 复杂切割: var arr=str.split(/正则/i)

c. 固定套路: var arr=str.split("")//将字符串打散为字符数组

2. RegExp对象:

(1). 创建:

a. 如果正则表达式是动态拼接而成的: var reg=new RegExp("正则","ig")

b. 如果正则表达式固定不变的: var reg=/正则/ig

(2). 验证格式:

var reg=/^正则$/;

var bool=reg.test(str)

(3). 既查找每个敏感词内容，又查找每个敏感词位置:

var reg=/正则/g

do{

var arr=reg.exec(str);

...

}while(arr!=null); //只有本次找到了敏感词才有必要继续执行下一次查找

3. Function

(1). 创建函数: 3种:

a. 用声明方式: 会被声明提前

function 函数名(形参变量列表){

函数体;

return 返回值

}

b. 用赋值方式: 不会被声明提前

var函数名=function (形参变量列表){

函数体;

return 返回值

}

c. 用new:

var 函数名=new Function("形参变量1","形参变量2",...,"函数体")

揭示了js中函数的本质:

function 其实是new Function()的简写

函数其实也是一个对象

函数名只是一个普通的变量而已

(2). 重载:

a. 何时: 一件事可能根据传入实参值的不同选择执行不同的逻辑时就要用重载

b. 如何:

1). 只定义一个函数，不要定义任何形参变量

2). 调用函数时，在函数内arguments对象会自动接住所有传入函数的实参值列表（类数组对象:有下标，有length，可遍历，但是不是数组家孩子）

3). 在函数内通过判断arguments的length属性不同或arguments中实参值不同来选择不同的逻辑执行

4). 如果想获得本次传入的实参值: arguments[下标]

(3). 匿名函数:

a. 何时: 如果一个函数只用一次时，不会重复使用，就要用匿名函数

b. 如何: 2种情况:

1). 回调函数: 主函数(function(形参变量){ ... })

2). 匿名函数自调: (function(){ ... })()

何时: 今后所有js代码都应该放在一个匿名函数自调中——避免使用全局变量

(4). 作用域和作用域链:

a. 作用域: js中包含两种作用域:

1). 全局作用域: window 始终在内存中，保存所有全局变量。全局变量特点：

i. 优点: 可重用

ii. 缺点: 随处可用——极易被污染

2). 函数作用域: 只在函数调用时才临时创建，函数调用后，函数作用域就释放。保存着函数内所有局部变量(2种: 形参和函数内var)。局部变量特点:

i. 优点: 仅函数内可用，不会被污染

ii. 缺点: 不可重用

b. 作用域链: 其实每个函数对象在创建时，都有一个属于自己的作用域链:

1). 保存了一个函数可用的所有作用域和变量

2). 控制了变量的使用顺序: 先函数内后全局——就近

(5). 闭包:

a. 何时: 想重用一个变量，但是又想保护变量不被篡改——其实就是给一个函数保护一个专属的变量，仅函数自己可反复用，别人无法使用。

b. 如何: 3步:

1). 用外层函数包裹内层函数和要保护的变量

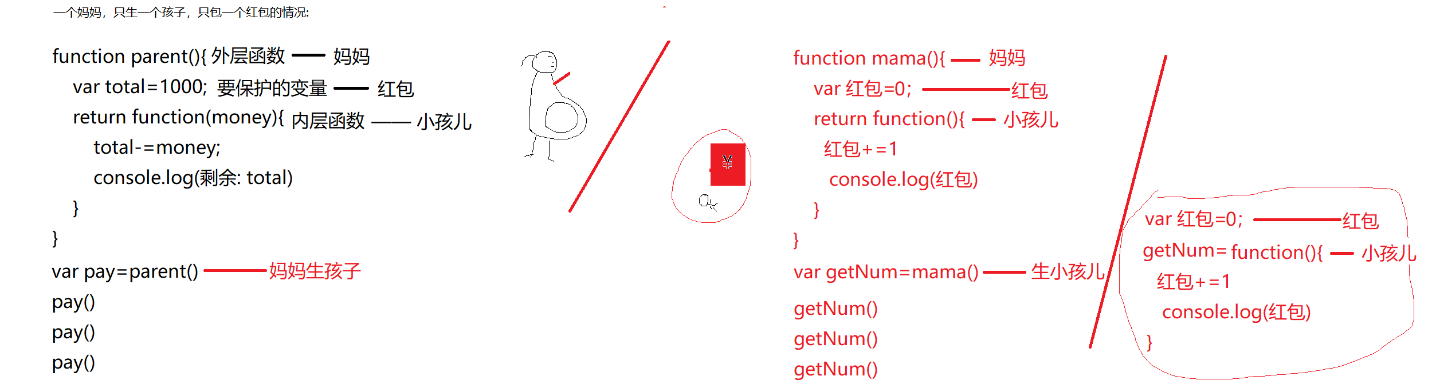
2). 外层函数将内层函数return到外层函数外部

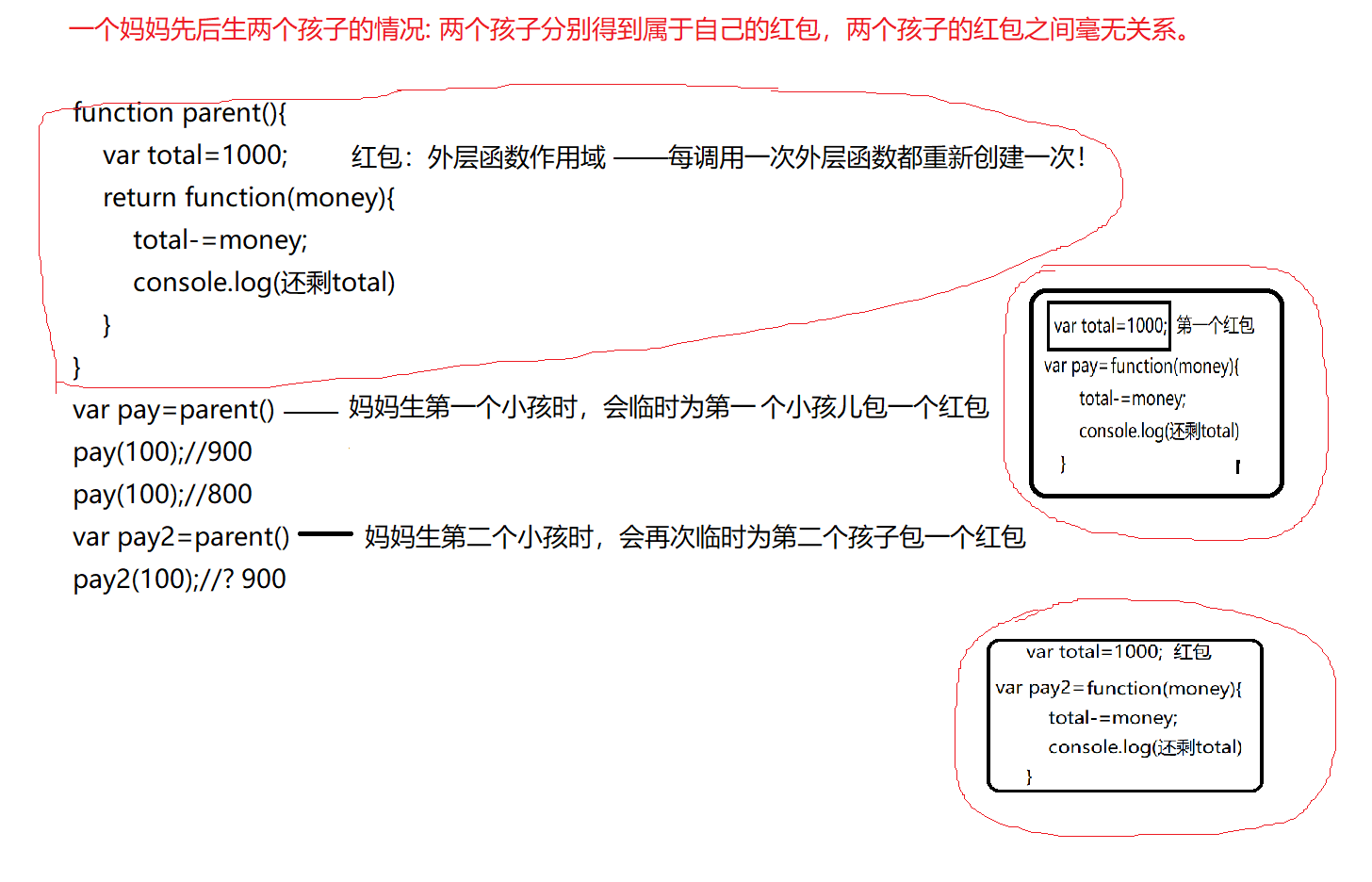
3). 外部的使用者需要调用外层函数，获得返回的内层函数对象，并将内层函数对象保存在变量中，反复使用。

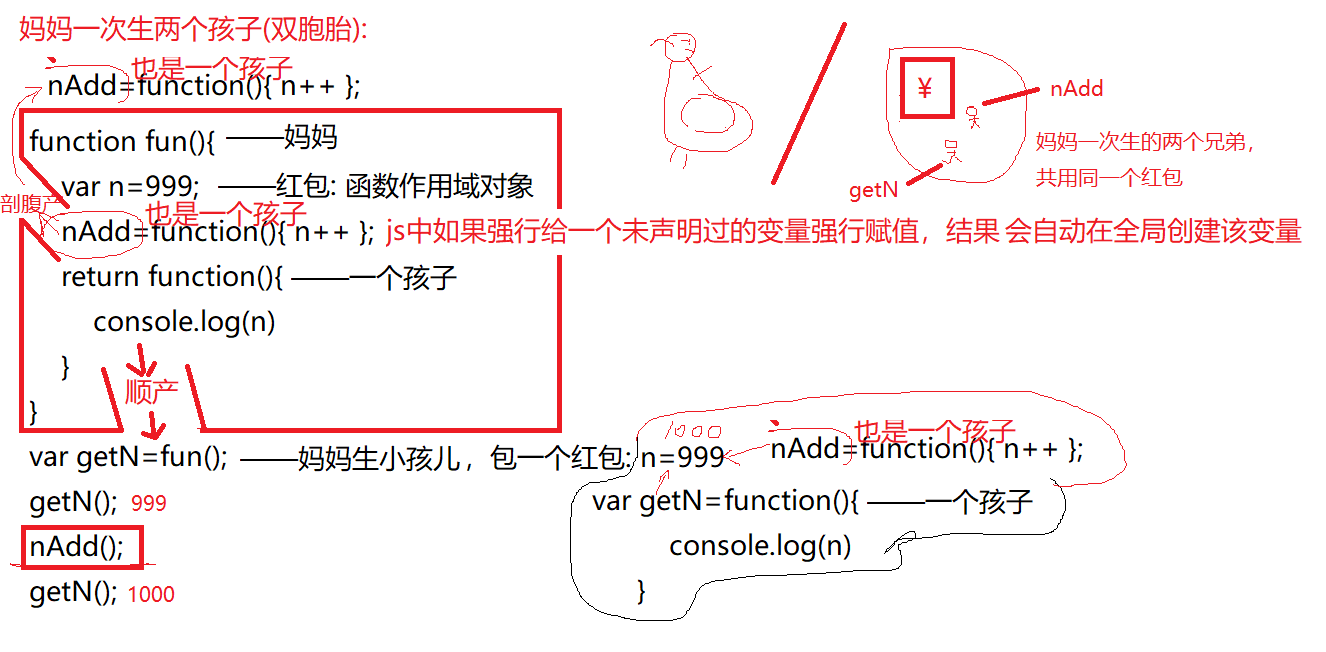
c. 闭包形成的原因: 外层函数调用后，外层函数的作用域对象被内层函数的作用域链引用着，无法释放，成为内层函数的私有财产，就形成了闭包。

d. 关于闭包的三个小故事：

对应小程序视频: 小程序->在线->JSCORE->day03 9. 关于闭包的比喻(三个小故事)







4. 面向对象:

(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()：2步 揭示: js对象底层都是关联数组

1). var 对象名=new Object()

2). 对象名.新属性=新值;

对象名.方法名=function(){ ... this.属性名 ...}

c. 用构造函数反复创建多个相同结构的对象: 2步

1). function 类型名(形参变量列表){

this.属性名=形参;

... = ... ;

~~this.方法名=function(){ ... this.属性名 ... }~~

}

2). var 对象名=new 类型名(实参值列表)

(2). 继承: 只要同一类型多个子对象都要使用的共有方法，应该集中定义在构造函数的原型对象中: 构造函数.prototype.共有方法=function(){ ... ... }

总结: this 判断this指谁，一定不要看this定义在哪儿！一定只看调用时.前是谁或有没有new.

1. obj.fun() this->obj

2. new Fun() this->new本次正在创建的一个新对象

3. Student.prototype.intr=function(){ ... } this->将来调用这个共有方法的.前的某个子对象

对应小程序视频:

小程序->在线->JSCORE->day03 9. 关于闭包的比喻(三个小故事)

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 constructor new

6. 继承 原型对象 prototype ...

作业:

1. 复习今日问题清单(小程序->首页->JSCORE->day04)，预习明日问题清单（小程序->首页->JSCORE->day05）

2. 看小程序视频学习自有属性和共有属性:

小程序->在线->JSCORE->day04 7. 自有属性，共有属性

3. 看小程序视频复习包装类型对象:

小程序->在线->JSCORE->day04 高频笔试题 包装类型 string

4. 看小程序视频学习高频笔试题：

小程序->在线->JSCORE->day04 高频笔试题 统计字符串中每种字符出现的次数

5. （学有余力）其余扩展视频，只要在三个月后鄙视面试前看完就行。