小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->JSCORE->day04、day05

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day04、day05

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

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频:

小程序->在线->JSCORE->day04

高频笔试题 9道

——通用解题方法：自己画图！

3. 单词列表: 小程序->单词->JSCORE->day04、day05

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. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

4. （这六天）必须用画图软件亲手画图！

如何提问: （vip待遇的剩余时间16天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 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:

\*\*\*\*面向对象\*\*\*\*

1. 封装

2. 继承

3. 多态

4. 自定义继承

\*\*\*\*ES5\*\*\*\*

1. 严格模式

\*\*\*\*面向对象\*\*\*\*

一. 封装: 3种方式

1. 用{}创建一个对象:

2. 用new Object()创建一个对象:

3. 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

(1). 问题: 如果需要反复创建多个相同结构的对象时，用{}代码就很冗余——重复。也不便于维护！

(2). 何时: 今后只要反复创建多个相同结构的对象时，都用构造函数来创建对象

(3). 如何: 2步

a. 先用构造函数定义同一类型多个对象的相同属性结构！

1). 什么是构造函数: 定义同一类型多个对象的相同属性结构的特殊函数

2). function 一类对象的统一类型名(形参1, 形参2,...){

//用this.属性名=xxx的方式，规定将来这类对象都必须有什么属性

//问题: 属性值不能写死的，因为每个对象虽然属性结构相同，但是属性值各不相同！所以，应该用形参变量来传入具体的属性值！

this.属性名1=形参1;

this.属性名2=形参2;

~~this.方法名=function(){ 将来构造函数中不应该包含方法定义！~~

~~...this.属性名...~~

~~}~~

}

b. 再用new反复调用构造函数创建多个相同结构的对象

1). var 对象名=new 类型名(属性值1, 属性值2, ...)

2). 鄙视: new做了几件事: 4件事

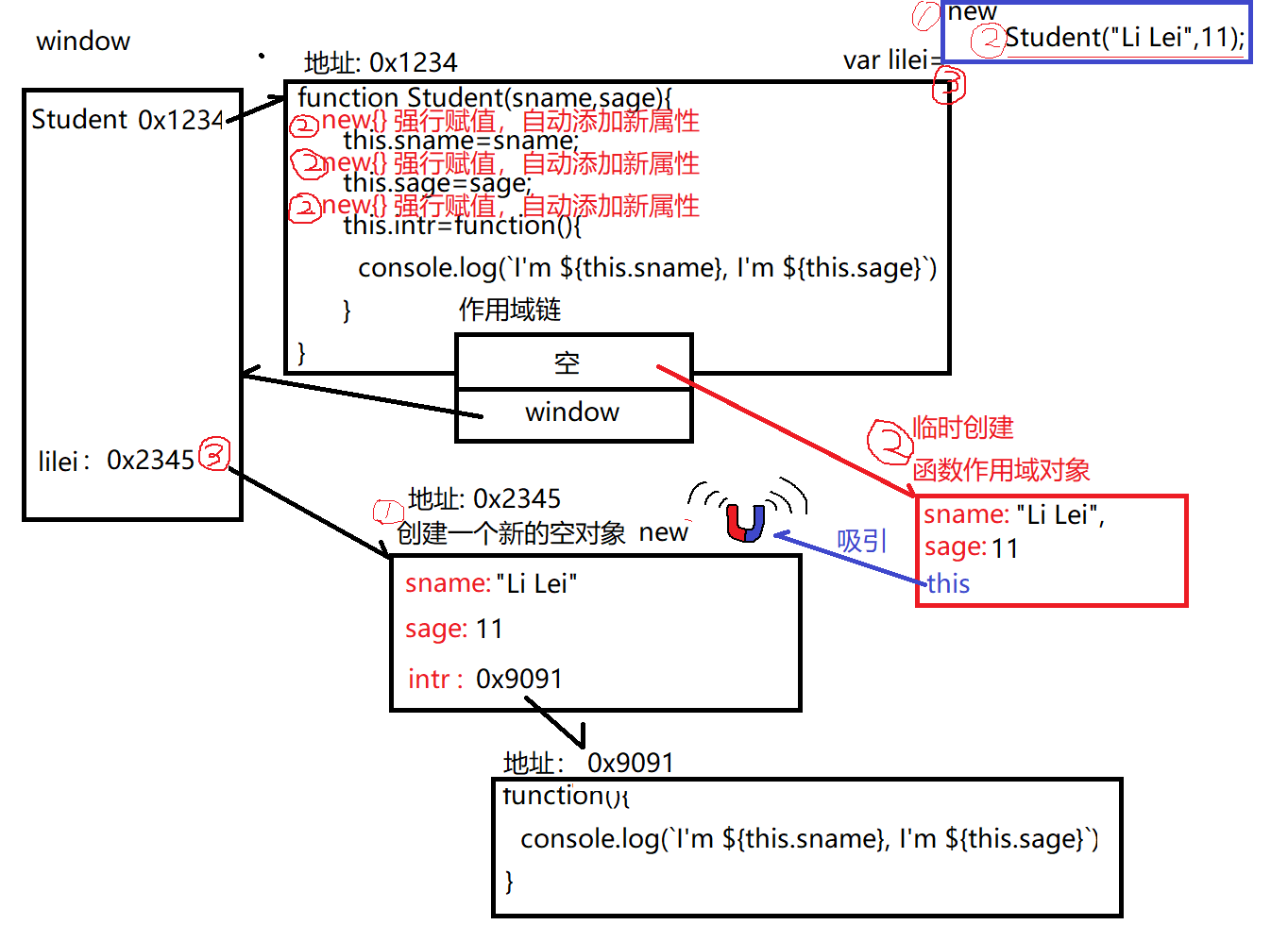
i. 创建一个新的空对象

ii. ?

iii. 调用构造函数，传入属性值实参，还自动将构造函数中的this全都指向正在创建的新对象！

结果: 构造函数中每一句话都相当于是给新对象强行赋值，自动添加新该属性。并将参数接住的属性值，保存进新对象的新属性中。

iv. new将新对象的地址保存到=左边的变量中(new有点儿return的作用)



(4). 示例: 定义学生类型的构造函数，反复创建两个学生对象:

1\_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>      //想创建两个学生对象: lilei和hmm      //定义所有学生类型的对象统一的属性结构:      //               形参变量永远可以改名，不必和属性名一致      //               但是强烈建议形参名和属性名保持一致      function Student(sname,sage){        //固定所有学生都有sname和sage两个属性，都要会做自我介绍intr()        this.sname=sname;        this.sage=sage;  ~~this.intr=function(){~~  ~~console.log(`I'm ${this.sname}, I'm ${this.sage}`)~~  ~~}~~      }      //想创建lilei      var lilei=new Student("Li Lei",11);      console.log(lilei);      //想创建hmm      var hmm=new Student("Han Meimei",12);      console.log(hmm);    </script>  </body>  </html>  运行结果:  Student {sname: "Li Lei", sage: 11, intr: ƒ}  Student {sname: "Han Meimei", sage: 12, intr: ƒ} |

(5). 构造函数的优点是: 重用对象结构代码

(6). 问题: 如果在构造函数中包含方法定义，则每创建一个对象，都会反复创建完全相同的函数对象的副本，而不是只创建一个所有对象共用！——浪费了内存！

但是确实所有对象都需要相同的方法！怎么办？

(7). 总结: 今后构造函数中就不应该包含方法定义

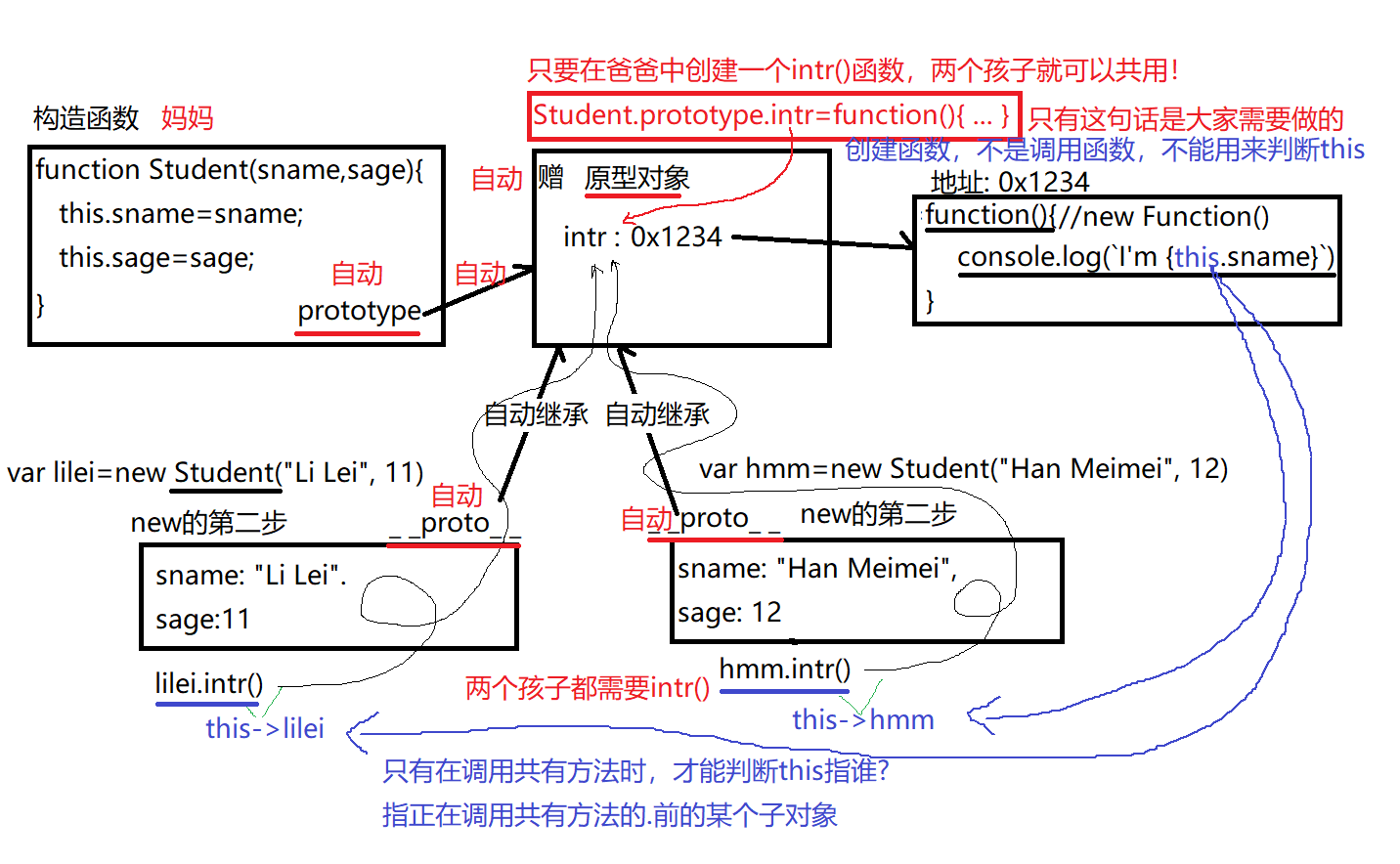
二. 继承:

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


1. 什么是继承: 父对象中的成员，子对象无需重复创建，就可直接使用！

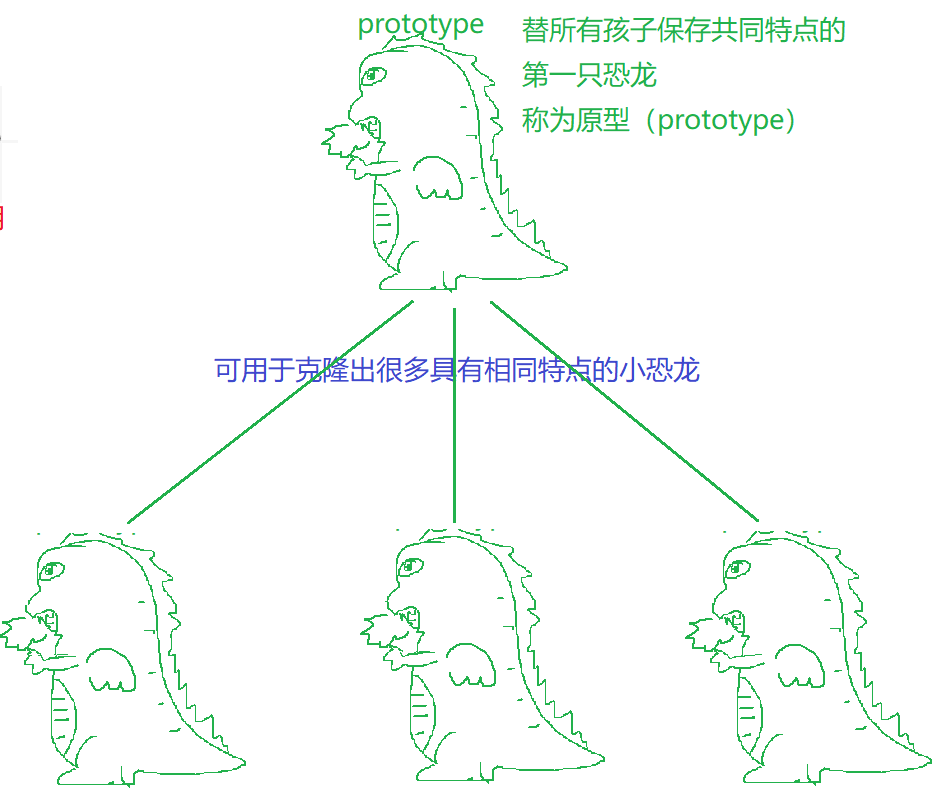
2. 何时: 同一类型的所有子对象，都需要相同的方法定义或属性值时，都会用继承的方式来实现。

3. 如何:



(1). (自动，不需要你手写！) 其实当我们定义一个构造函数时，js程序都会自动赠送我们一个原型对象(prototype)，也称为该类型下所有孩子的父对象。

(2). 什么是原型对象(prototype)：替所有子对象保存共有方法和属性值的父对象



(3). (自动，不需要你手写！) 当new一个新对象时，new的第二步是自动让新子对象继承构造函数的原型对象。自动设置子对象的\_ \_proto\_ \_属性指向构造函数的原型对象。

(4). 结果，将来用子对象试图访问一个子对象没有的共有方法或属性时，js程序先在子对象自身内部查找。如果子对象内部没有，js程序会自动延\_ \_proto\_ \_向父对象(原型对象)中查找是否包含要用的方法或属性。只要在父对象（原型对象）中找到想要的方法，依然可以用"子对象.共有方法名()"调用。看起来就像子对象在用自己的方法一样！

(5). 问题: 构造函数的原型对象开始时是空的！如何向原型对象中添加共有属性和方法？——唯一的办法只有“强行赋值”！

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

4. 总结: 今后，只要所有子对象都要共有的方法或属性值，都要集中保存在构造函数的原型对象中！

5. 问题: 为什么指同一个原型对象，从构造函数访问时，要用prototype属性，而从子对象访问时却要用\_ \_proto\_ \_。构造函数的prototype属性和子对象的\_ \_proto\_ \_指的到底是不是同一个对象呢？

答: prototype和\_ \_proto\_ \_，指的是同一个对象。只不过，通过不同辈分的其它对象访问同一个原型对象时，称呼不同！

6. 问题: 原型对象中的this指谁？

(1). 错误: ~~因为构造函数.prototype.共有方法=function(){ ... }~~

~~所以共有方法中的this指.前的原型对象！~~

因为: 这句话只是向原型对象中添加函数，并不是调用共有方法。

(2). 正确: 判断this不要看定义在哪儿，应该看何时被谁调用！

因为共有方法迟早会被"子对象.共有方法()"调用。所以，共有方法中的this不确定。但是肯定指将来正在调用这个共有方法的.前的某个子对象。

7. 示例: 为所有学生的原型对象中添加共用的自我介绍方法

2\_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>      //构造函数      function Student(sname,sage){        this.sname=sname;        this.sage=sage;      }      //因为今后所有学生对象都要能自我介绍，所以，应该在学生类型的构造函数的原型对象中强行添加一个intr()方法。供将来所有学生共用！      Student.prototype.intr=function(){        console.log(`I'm ${this.sname}, I'm ${this.sage}`)      }      console.dir(Student);      //想创建lilei      var lilei=new Student("Li Lei",11);      //想创建hmm      var hmm=new Student("Han Meimei",12);        console.log(lilei);      console.log(hmm);      lilei.intr();      hmm.intr();      //亲自鉴定:      //判断李磊的爹是不是构造函数妈妈的原型对象      console.log(lilei.\_\_proto\_\_==Student.prototype);//true      //判断hmm的爹是不是构造函数妈妈的原型对象      console.log(hmm.\_\_proto\_\_==Student.prototype);//true      //判断lilei的爹和hmm的爹是不是同一个人      console.log(lilei.\_\_proto\_\_==hmm.\_\_proto\_\_);//true    </script>  </body>  </html>  运行结果: |

8. 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). 什么是:

a. 自有属性: 保存在当前对象内部，仅归当前对象自己所有的属性

b. 共有属性: 保存在原型对象(父对象)中，归多个子对象共有的属性

(2). 获取属性值: 无论自有属性，还是共有属性，都可用"子对象.属性名"，用法是相同的！

(3). 修改属性值:

a. 自有属性: 应该用子对象直接修改。子对象.自有属性名=属性值

b. 共有属性:

1). ~~错误: 用子对象.共有属性方式=新值~~

因为: 共有属性归所有子对象共用，如果允许某一个子对象擅自篡改，则牵一发而动全身！——危险！

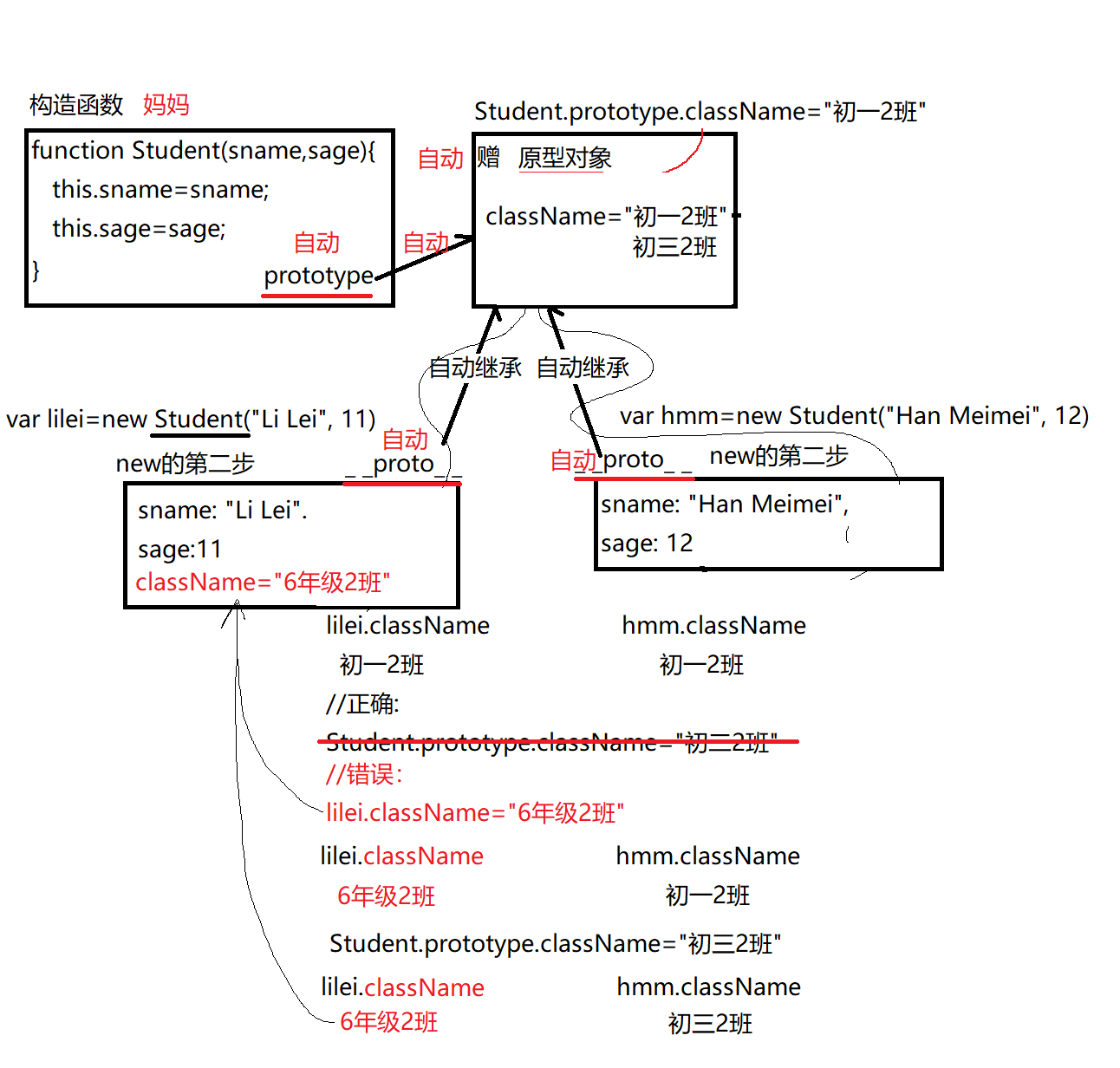
所以: js禁止子对象修改原型对象中的共有属性

但是: 如果强行用子对象.共有属性=新值，js程序会自动给这个子对象添加一个自有的同名属性保存新属性值。而不会影响原型对象中的共有属性

结果: 从此，这个子对象，在这个共有属性的使用上，与其他子对象，分道扬镳！

2). 正确: 共有属性，只能用原型对象修改: 构造函数.prototype.共有属性=新值

(4). 示例: 为李磊和hmm添加共有属性className，存储二人共同的班级名，并用正确和错误的方式分别修改共有属性className，查看结果:



3\_property.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>      //构造函数      function Student(sname,sage){        this.sname=sname;        this.sage=sage;      }      //所有学生都来自初一2班，所以应该在原型对象中添加一个className属性保存所有学生统一的班级      Student.prototype.className="初一2班";      //创建lilei      var lilei=new Student("Li Lei",11);      //创建hmm      var hmm=new Student("Han Meimei",12);      console.log(lilei);      console.log(hmm);      //读取李磊的姓名和班级      console.log(lilei.sname, lilei.className);      //读取hmm的姓名和班级      console.log(hmm.sname, hmm.className);      //李磊想改名叫李晓磊      lilei.sname="Li Xiaolei";      console.log(lilei);      //第一年，输出两个人的班级      console.log(lilei.className, hmm.className);      //过了一年，两个人同时升级到初二2班      //错误: 用某个一个子对象，试图修改原型对象中的共有属性值      lilei.className="6年级2班";      //正确:      //Student.prototype.className="初二2班";      //再输出两个人的班级，希望两人都是初二2班      console.log(lilei.className, hmm.className);      //又过了一年，大家一起升初三2班      Student.prototype.className="初三2班";      console.log(lilei.className, hmm.className);    </script>  </body>  </html>  运行结果：  Student {sname: "Li Lei", sage: 11}  Student {sname: "Han Meimei", sage: 12}  Li Lei 初一2班  Han Meimei 初一2班  Student {sname: "Li Xiaolei", sage: 11}  初一2班 初一2班  6年级2班 初一2班  6年级2班 初三2班 |

9. 内置对象的原型对象:

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

(1). 什么是内置类型/内置对象: ECMAScript标准中规定的，浏览器已经实现的对象或类型。

(2). 11种:

String Number Boolean

Array Date RegExp Math(不是类型，已经是一个对象了)

Error

Function Object

global(在浏览器中被window代替)

(3). 什么是类型: 构造函数+原型对象形成的整体

a. 构造函数负责: 创建该类型的子对象。构造函数中的所有属性会成为将来子对象内的自有属性

b. 原型对象负责: 集中保存该类型所有子对象公共的成员（属性值和方法）。原型对象中的方法和属性值都会成为将来所有子对象的共有成员。

(4). 问题: 我怎么知道数组家共有哪些函数？

只要看类型.prototype中有哪些函数即可！

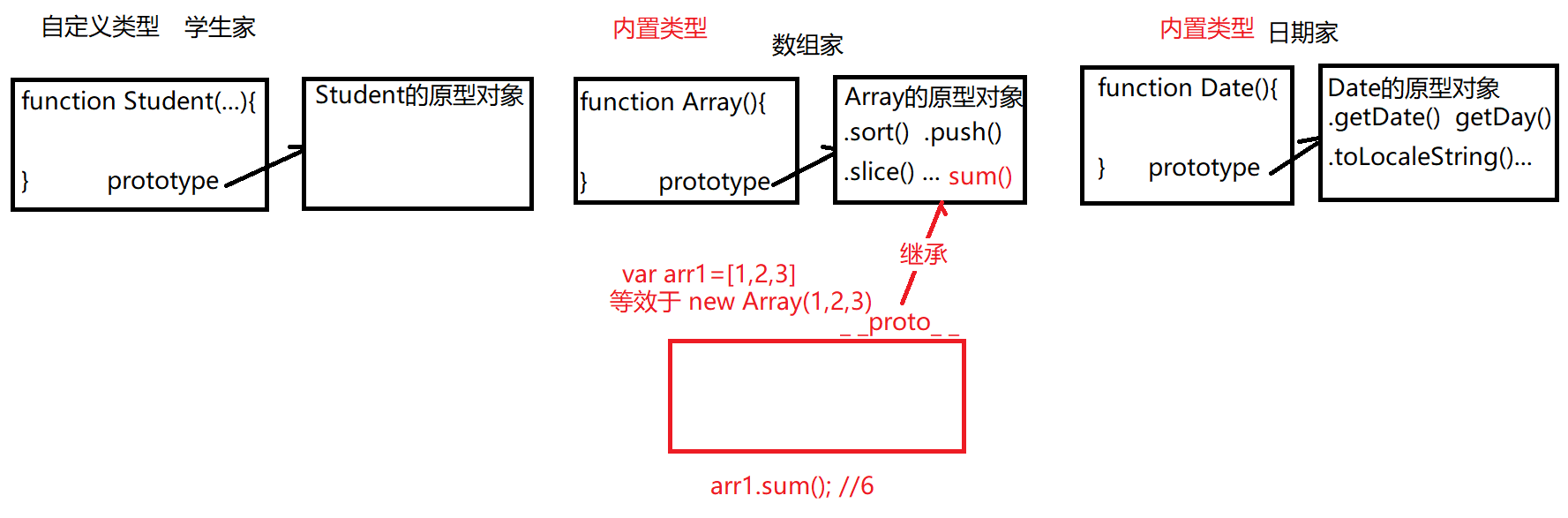
(5). 问题: 如果经常对数组执行一种操作，但是原型对象中没有提供现成的函数可用，怎么办？——只要自定义一个函数，强行添加到数组的原型对象中即可!

Array.prototype.新函数名=function(){

//this->今后调用这个函数的.前的某个子对象

}

结果: 将来任何一个数组都可以.新函数名()



(6). 示例: 为数组类型添加一个对所有元素求和的共有函数

4\_Array.prototype.sum.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>      //项目中，经常需要对各种数组内容求和      //应该自定义一个sum函数，强行添加到数组类型的原型对象中      Array.prototype.sum=function(){        console.log(`调用我自己定义的sum()函数！`);        //this->将来调用这个sum()函数的.前的某个数组        //比如，将来arr1.sum()，this->arr1        //     将来arr2.sum(), this->arr2        //对当前正在调用sum()的某个数组中所有元素求和        var result=0;        //遍历数组中每个元素        for(var i=0;i<this.length;i++){          result+=this[i]        }  //错误: in会把sum()函数本身的定义也遍历进来        // for(var key in this){        //   result+=this[key]        // }        return result;      }      console.log(Array.prototype);        //希望:      var arr1=[1,2,3];      var arr2=[1,2,3,4,5];      console.log(arr1.sum())//6; //报错: sum不是一个函数      console.log(arr2.sum())//15    </script>  </body>  </html>  运行结果:  [sum: ƒ, constructor: ƒ, concat: ƒ, ...]  调用我自己定义的sum()函数！  6  调用我自己定义的sum()函数！  15 |

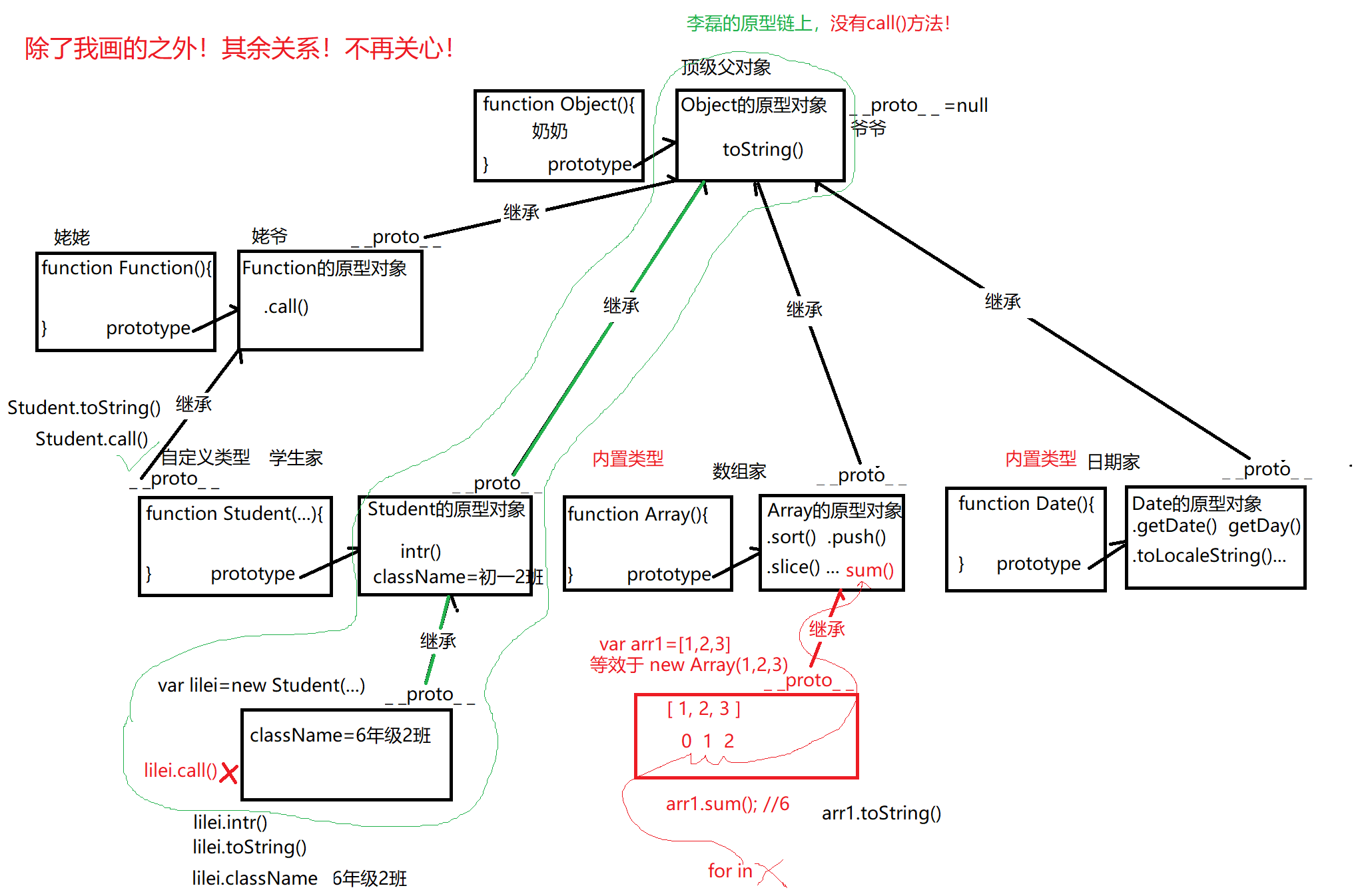
10. 原型链:

对应小程序视频: 小程序->在线->JSCORE->day04 9. 原型链prototype chain

(1). 什么是原型链: 由多级父对象，逐级继承，形成的链式结果

(2). 原型链保存着一个对象可用的所有属性和方法

(3). 原型链控制着属性和方法的使用顺序：先自有，再共有——就近原则



(4). 问题: 不同类型的子对象，调用toString()，输出的结果格式千差万别！

三. 多态:

1. 什么是多态: 一个函数，不同情况下，表现出不同的状态！

2. 其实多态分两种情况:

(1). 重载(overload): 一个函数，根据传入的实参值不同，执行不同的逻辑！(已学过)

(2). 重写(override):

推翻

3. 什么是重写: 在子对象中定义一个和父对象中成员同名的成员！

4. 为什么: 因为从爹继承来的东西，不总是好用的！

5. 何时重写: 今后，只要从爹继承来的东西不好用！就用重写自己的！

6. 如何: 只要在当前子对象中定义一个和父对象中不好用的成员同名的新成员即可。

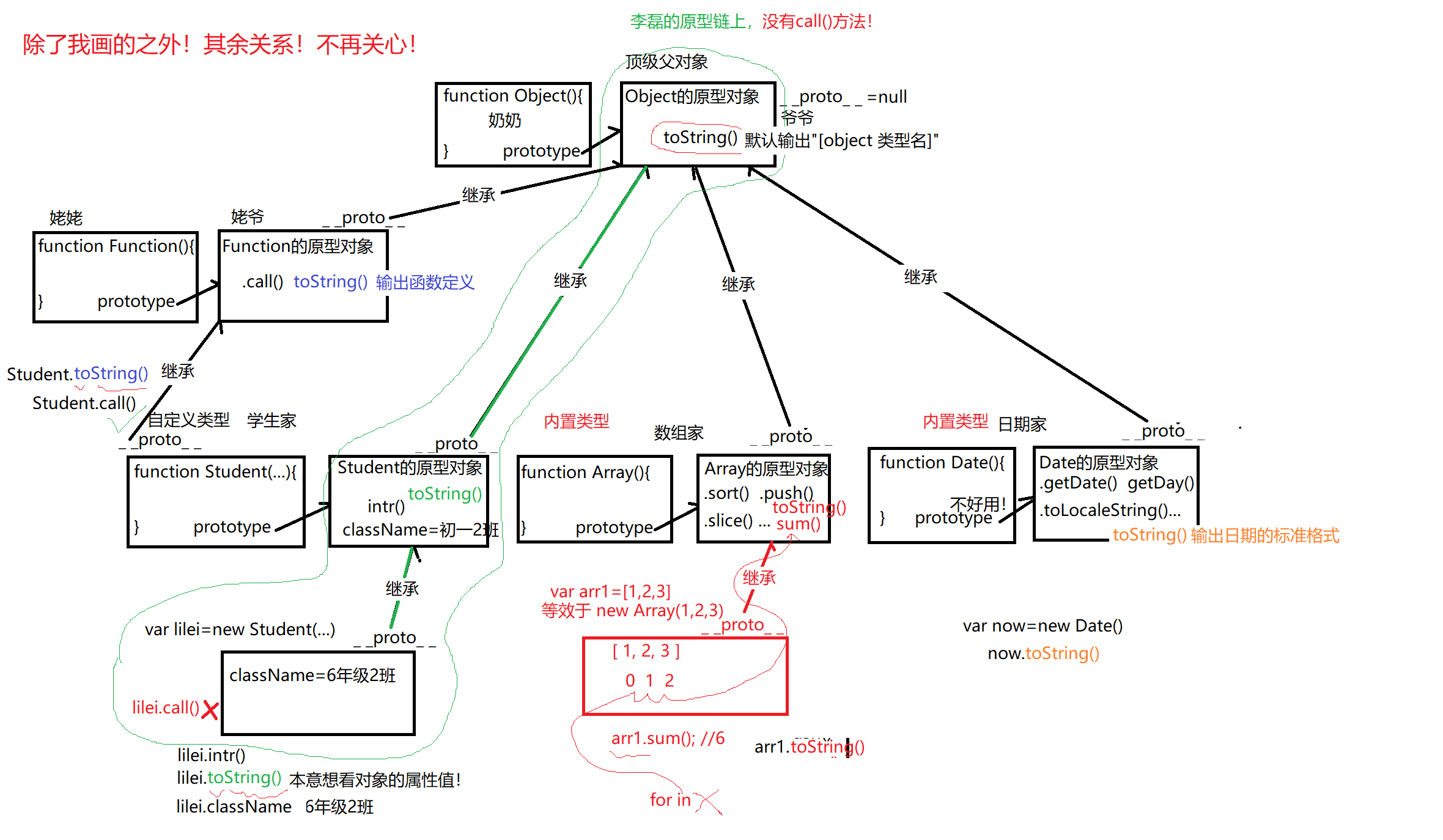
7. 结果: 根据就近原则，优先使用子对象自己定义的。不再使用父对象中不好用——推翻

8. 问题: 不同类型的子对象，调用toString()，输出的结果格式千差万别？

答: 所有内置类型的原型对象中，为了自己家孩子着想，已经重写了适合自己家孩子的toString()

9. 问题: 但是，咱们自己定义的类型中还没有重写toString()方法，所以非常不便于调试对象的内容！

解决: 强烈建议所有自定义类型和自定义对象，都要重写自定义的toString()方法，便于调试！



10. 示例: 为自定义类型重写toString()方法

5\_override.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>      function Student(sname,sage){        this.sname=sname;        this.sage=sage;      }      //为了让Student家的孩子们，也有好用的toString()，需要在Student的原型对象中，添加一个自定义的好用的toString()      Student.prototype.toString=function(){        //字符串格式和内容可以自定义        return `{          学生姓名: ${this.sname},          学生年龄: ${this.sage}        }`      }      var lilei=new Student("Li Lei",11);      var arr1=new Array(1,2,3);      var now=new Date();      //分别输出函数，lilei，arr1和now的toString()      console.log(Student.toString());      console.log(lilei.toString());      console.log(arr1.toString());      console.log(now.toString());    </script>  </body>  </html>  运行结果:  function Student(sname,sage){  this.sname=sname;  this.sage=sage;  }  {  学生姓名: Li Lei,  学生年龄: 11  }  1,2,3  Fri Jul 03 2020 18:11:06 GMT+0800 (中国标准时间) |

|  |
| --- |
| 面向对象总结:  1. 封装: 先创建对象: 2种:  (1). 如果只创建一个对象: {}  (2). 如果反复创建多个相同结构的对象: 构造函数  2. 继承: 所有子对象共用的方法和属性值，必须集中定义在原型对象中  3. 多态: 如果父对象继承来的成员不好用！可以重写自己的！ |

|  |
| --- |
| 补: for in 循环，~~禁止用于遍历索引数组~~！只能遍历关联数组或对象  因为: in不但遍历当前对象的每个属性（包括数字下标和非数字下标），且会自动延\_ \_proto\_ \_继续遍历父对象中的未隐藏的成员！ |

总结:

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(){ ... }，底层等效于 var 函数名=new Function(...) |

(2). 重载: 今后，只要一个函数将来调用时不确定传入实参值的个数，都要用重载

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（scope）:

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(2种:形参和函数内var)

优: 仅函数内可用，不会被污染，缺: 不可重用

函数作用域对象原理:

i. 函数作用域对象只在调用函数时才临时创建，保存函数内的局部变量

ii. 当函数调用过后，函数作用域对象和局部变量就释放了！

b. 作用域链(scope chain/scopes): 保存一个函数可用的所有作用域对象的链式结果(好友列表)学名就叫作用域链。作用域链保存着一个函数可用的所有变量，作用域链控制着变量的使用顺序——先局部后全局。

查看一个函数在未调用时的作用域链: console.dir(函数名); //展开看到[[scopes]]

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

a. 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——套娃

2). 外层函数妈妈用return把内层函数孩子返回到外部——生娃

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。——接生

b. 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——(孩子拿着妈妈给的红包)自立门户

c. 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: 变量=null

d. 画简图:

1). 找三样东西:

i. 外层函数——妈妈

ii. 受保护的变量——红包

iii. 外层函数共返回几个内层函数对象——孩子(们)

2). 规律:

i. 调用一次妈妈，返回的多个孩子，共用一个红包

ii. 调用多次妈妈，返回的多个孩子，有各自的红包，互不干扰！

4. 面向对象: 封装， 继承， 多态

(1). 封装: 3种:

a. 用{}只创建一个对象:

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。但是如果属性名来自于变量就只能用[]

iii. 访问不存在的属性，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，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！

v. 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. (待续)

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: String, Number, Boolean, Array, Date, RegExp, Math, Error, Function, Object, global

2). 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(){ ... }

d. 原型链: 保存着一个对象可用的所有属性和方法。控制着属性和方法的使用顺序：先自有再共有——就近原则！

(3). 多态: 重点讲重写：如果子对象觉得从父对象继承来的成员不好用，可以在子对象自己内部重写和父对象同名的成员，覆盖父对象的成员，优先使用自己的。

|  |
| --- |
| 面向对象总结:  1. 封装: 先创建对象: 2种:  (1). 如果只创建一个对象: {}  (2). 如果反复创建多个相同结构的对象: 构造函数  2. 继承: 所有子对象共用的方法和属性值，必须集中定义在原型对象中  3. 多态: 如果父对象继承来的成员不好用！可以重写自己的！ |

|  |
| --- |
| 总结: this 4种: 判断this，~~一定不要看定义在哪儿！~~只看调用时！  1. obj.fun() this->obj  2. fun() 或 (function(){ ... })() this->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->将来谁调用指谁，同第一种情况 |

今日对应小程序视频:

小程序->在线->JSCORE->day04 5. 构造函数 constructor new

6. 继承 原型对象 prototype ...

7. 自有属性，共有属性

8. 11种内置类型/内置对象...

9. 原型链prototype chain

小程序->在线->JSCORE->day05 1. 多态override toString()

作业:

1. 复习今日小程序问题清单: 小程序->首页->JSCORE->day04、day05

2. 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 包装类型String

3. 看小程序视频预习:

小程序->在线->JSCORE->day05 3. ES5严格模式

4. 保护对象属性 开关

5. 访问器属性——难