提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->JSCORE->day04里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->JSCORE->day04里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线->JSCORE->day04里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

衡量是否学会的标准：

1. 可以用自己的话把注释的步骤写出来！才算会！（函数名或程序忘了都没关系！）

2. 可以自己画出内存变化过程图！才算会！（凡是只用眼睛看想得出揭露的！就都是错误的结论！）

不要去网上乱找案例看！只看我的视频！

正课:

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. 什么是面向对象: 程序中先用对象保存现实中一个事物的属性和功能，然后再按需访问对象中保存的属性和功能

2. 为什么: 便于大量数据的维护和使用

3. 何时: 今后所有程序都使用面向对象思想开发出来的！

4. 如何: 面向对象三大特点: 封装，继承，多态

二. 封装:

1. 什么是: 其实就是创建一个对象，集中保存现实中一个事物的属性和功能

2. 为什么: 便于大量数据的维护和使用

3. 何时: 今后只要使用面向对象方式编程，都要先封装对象，再按需访问对象中的成员

4. 如何创建对象: 3种方式:

(1). 用{}直接量创建对象:

a. var 对象名={

属性名: 属性值,

... : ... ,

方法名: function(){

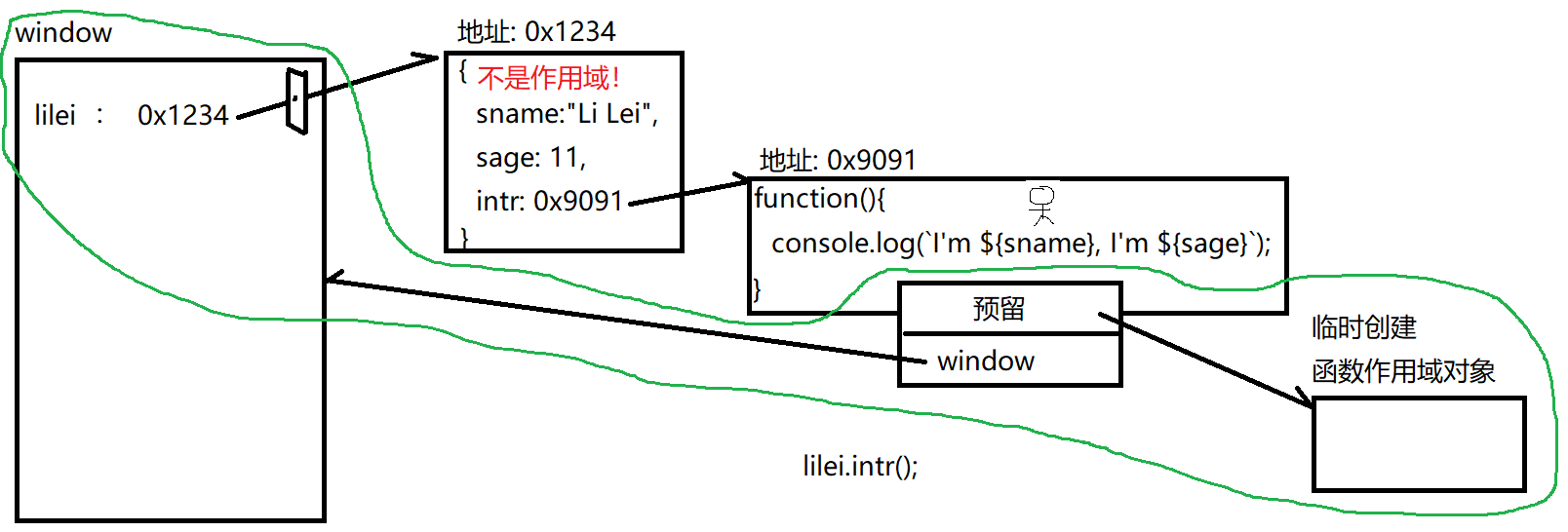
... this.属性名 ...

}

}

b. 问题: 对象自己的函数，想访问对象自己的属性，竟然报错！xxx is not defined

因为: 所有不加.的变量，默认只能在函数作用域和全局作用域window中查找变量。无权进入对象中查找属性。

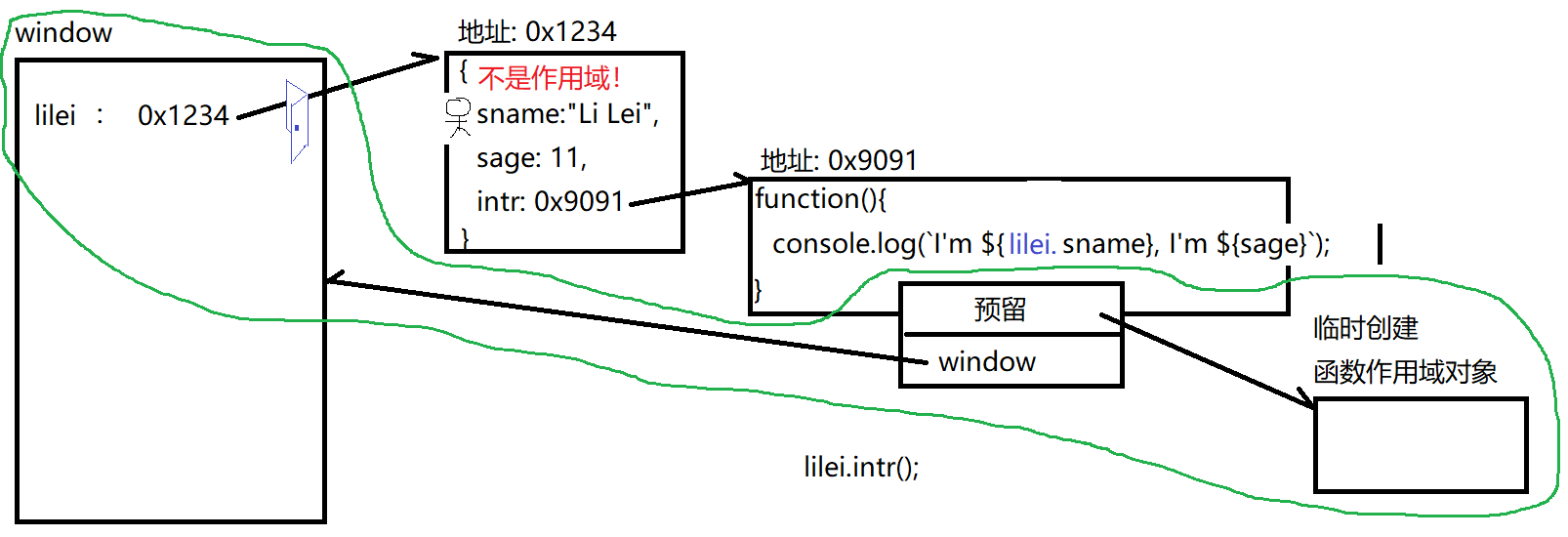


c. 解决:

1). 不好的解决: 在对象方法内写死"对象名.属性名"，来指引js引擎先找到对象，再进入对象中查找属性

问题: 紧耦合。万一对象名修改了，就要被迫跟着修改方法中的写死的对象名。一旦忘记修改方法中写死的对象名，依然会出错！

希望: 松耦合。即使外部对象名修改了，方法内也不需要做任何修改，依然可以访问对象中的属性！



2). 好的解决: 用this关键词代替写死的对象名: "this.属性名"

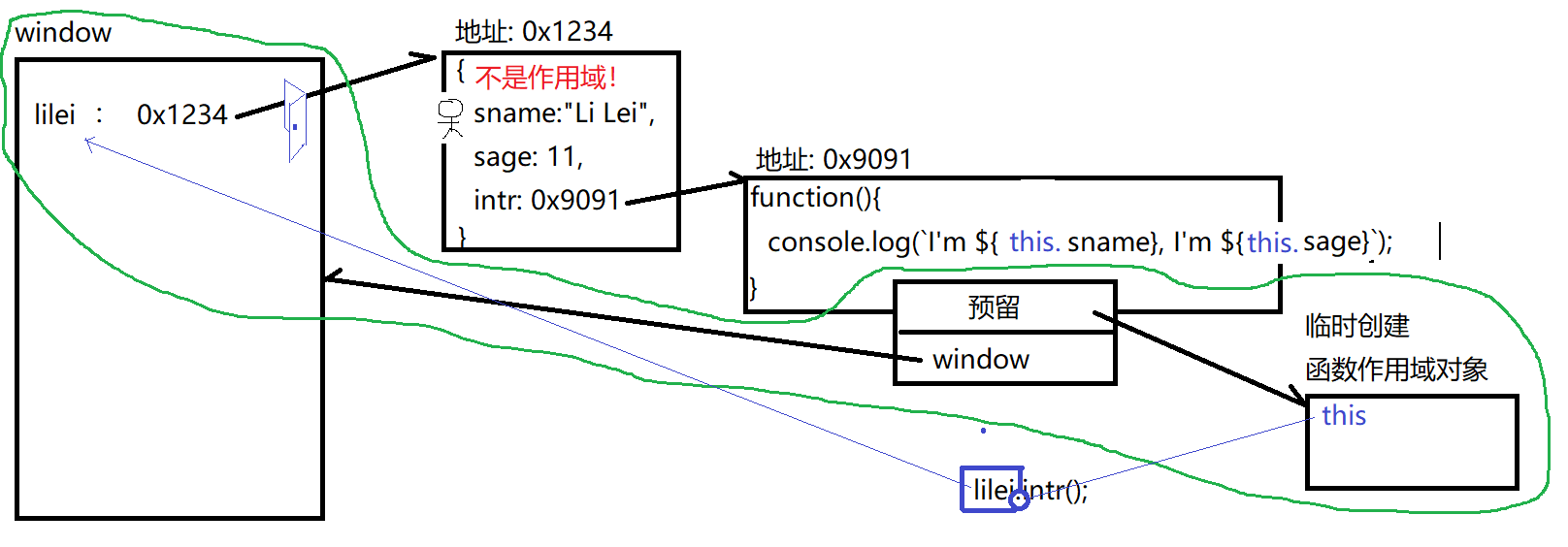
i. 什么是this:

每个函数内自带的，

可以自动获得正在调用当前函数的.前的对象

的关键词

ii. 何时: 只要对象自己的方法内，想使用对象自己的属性，必须加"this.属性名"



|  |
| --- |
| 方法 vs 函数  相同: 都是function！  不同: 独立于任何对象之外，单独存在的function，称为函数  保存在对象内的function，才称为方法！ |

d. 示例: 定义对象保存了李磊的属性和功能

|  |
| --- |
| //想创建一个对象lilei，保存李磊的两个属性: 学生姓名、学生年龄和一个李磊会做的事儿:自我介绍  var lilei={    sname:"Li Lei",    sage: 11,    intr:function(){      console.log(`I'm ${this.sname}, I'm ${this.sage}`)    }  }  //想获取李磊的年龄  console.log(lilei.sage);  //想请李磊做个自我介绍  lilei.intr();  //过了一年，李磊长了一岁  lilei.sage++;  //再获取李磊的年龄  console.log(lilei.sage);  //再请李磊做个自我介绍  lilei.intr(); |

(2). 用new Object()创建:

a. 2步:

1). 想用new Object()创建一个空对象

var lilei=new Object(); //{}其实就是new Object的简写！

2). 为空对象强行添加新属性

lilei.sname="Li Lei";

lilei.sage=11

lilei.intr=function(){

... this.sname ...

}

b. 揭示了: js中一切对象低层都是关联数组！

1). 和关联数组一样，对象内部保存的也是"名值对儿"的集合

2). 都能用下标方式访问对象中的成员属性，且都能用.简写下标访问:

lilei["sage"] 可简写为 lilei.sage

lilei.sage 会被自动翻译为 lilei["sage"]

强调: 如果要访问的属性名不是固定的，而是来自于其他变量或需要动态拼接，则~~不能用.~~，只能用[]，还不能加""

for(var key in lilei){

//key变量接住的值每一轮都在变化: sname sage intr className

console.log(`${key} : ${lilei[key]}`)

~~//console.log(`${key} : ${lilei["key"]}`) //变量不能加""~~

~~//console.log(`${key} : ${lilei.key}`) //.key会被自动翻译为["key"]~~

}

3). 关联数组可通过强行赋值方式随时添加新元素，对象也可通过强行赋值方式随时添加新属性！而不会报错！

4). 访问关联数组中一个不存在的位置，不会报错，而是返回undefined。

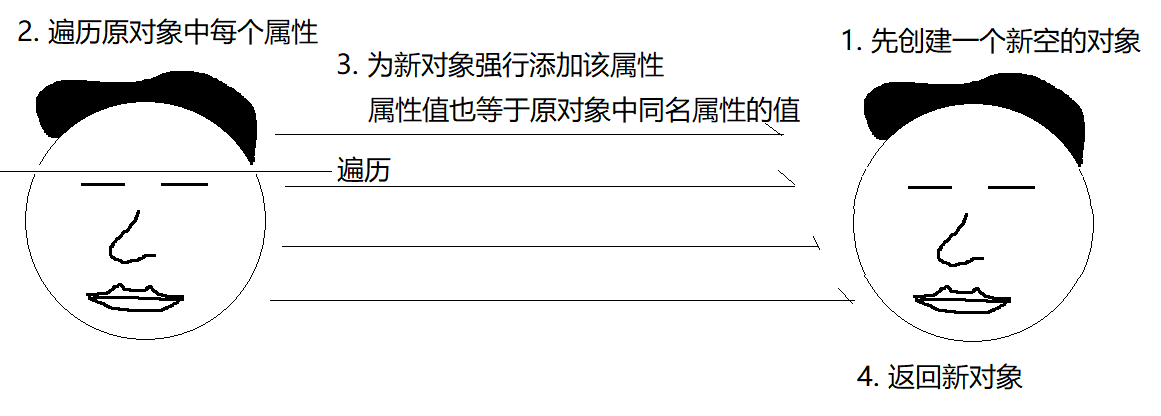
同样，访问对象中一个不存在的属性，也不会报错！也返回undefined

5). 都能用for in遍历每个成员！

c. 示例: 比较关联数组和对象

|  |
| --- |
| var lilei={    sname:"Li Lei",    sage: 11  }  console.log(lilei);  //想获取李磊的年龄  console.log(lilei["sage"]);  //尝试为李磊强行添加一个新属性  lilei.className="初一2班";  //试图访问李磊的爱好  console.log(lilei.fav);//undefined  //遍历lilei中每个成员  for(var key in lilei){    //每一轮都在变化: sname  sage   intr    className    console.log(`${key} : ${lilei[key]}`)  //console.log(`${key} : ${lilei["key"]}`) //变量不能加""  //console.log(`${key} : ${lilei.key}`)  //                             .key会被自动翻译为["key"]  }  var ym=[];  ym["math"]=89;  ym.chs=69;  ym["eng"]=91;  console.log(ym);  //试图访问杨幂的className  console.log(ym["className"])//undefined  //强行为ym添加一个新属性，爱好:跑步  ym.fav="跑步";  //遍历ym中每个成员  for(var key in ym){    //in 依次取出对象中每个属性名,将属性名保存到in前的变量中    //变量key每次循环会依次得到: math  chs  eng   fav    //又因为对象或关联数组中的属性名，有一个技术学名叫"键"    //属性值，简称为值    //所以有人也称对象和关联数组为"键值对儿"的集合    //而键英文就是key    //所以key可以用来表示属性名    console.log(`${key} : ${ym[key]}`)  } |

d. 示例: 克隆一个对象



|  |
| --- |
| var lilei={    sname:"Li Lei",    sage:11  }  //定义一个函数，接受一个旧对象，克隆出一个一模一样的新对象  function clone(oldObj){    //1. 先创建一个新的空对象，准备接收旧对象中的属性和属性值    var newObj={};//=new Object()    //2. 遍历旧对象中每个属性和属性值    for(var key in oldObj){      //3. 每遍历旧对象中一个属性，就为新对象强行赋值添加一个同名的新属性，属性值也等于旧对象中当前同名属性的值      newObj[key]=oldObj[key];    }    //4. 克隆完毕，返回新对象    return newObj;  }  var lilei2=clone(lilei);  console.log(lilei);  console.log(lilei2);  console.log(lilei==lilei2);  //lilei2挣了100块钱  lilei2.money=100;  console.log(lilei.money);  console.log(lilei2.money); |

(3). 用构造函数反复创建多个相同结构的对象

a. 问题: 前两种方式一次只能创建一个对象。如果要反复创建多个相同结构的对象时，代码会很繁琐，且不便于维护

b. 何时: 今后只要反复创建相同类型的多个对象时，都用构造函数

c. 如何: 2步:

1). 定义一个构造函数来描述一类对象统一的结构

i. function 类型名(形参1, 形参2,...){

this.属性名=形参1;

this.属性名=形参2;

~~this.方法名=function(){~~

~~... this.属性名 ...~~

~~}~~

}

ii. 强调:

① 希望将来出现在新对象中的每个属性，都要用"this."定义

② 将来的新对象规定有几个属性，构造函数上就要定义几个形参变量，用来在将来创建新对象时，接受新对象的具体属性值。

2). 反复用new 调用构造函数，反复创建相同类型的多个对象

i. var 新对象=new 类型名(属性值1, 属性值2,...)

ii. 结果:

① new先创建一个空对象

②调用构造函数时的实参值会传给构造函数的形参变量

③构造函数中规定的所有属性，都会被添加到新对象内

④ new会返回新创建的对象，保存到=左边的变量中

d. 示例: 定义描述学生类型对象统一结构的构造函数，并用构造函数创建两个学生对象

|  |
| --- |
| //先创建一个学生对象保存李磊的属性和功能  //又要创建一个学生对象保存韩梅梅的属性和功能  //因为李磊和韩梅梅都是学生（Student）类型的对象，所以属性结构和方法定义完全相同  //所以应该用构造函数来定义  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调用构造函数创建lilei对象  var lilei=new Student("Li Lei",11);  //用new调用构造函数创建hmm对象  var hmm=new Student("Han Meimei",12);  console.log(lilei);  console.log(hmm); |

e. new的原理: new做了4件事

1). 先创建一个新的空对象

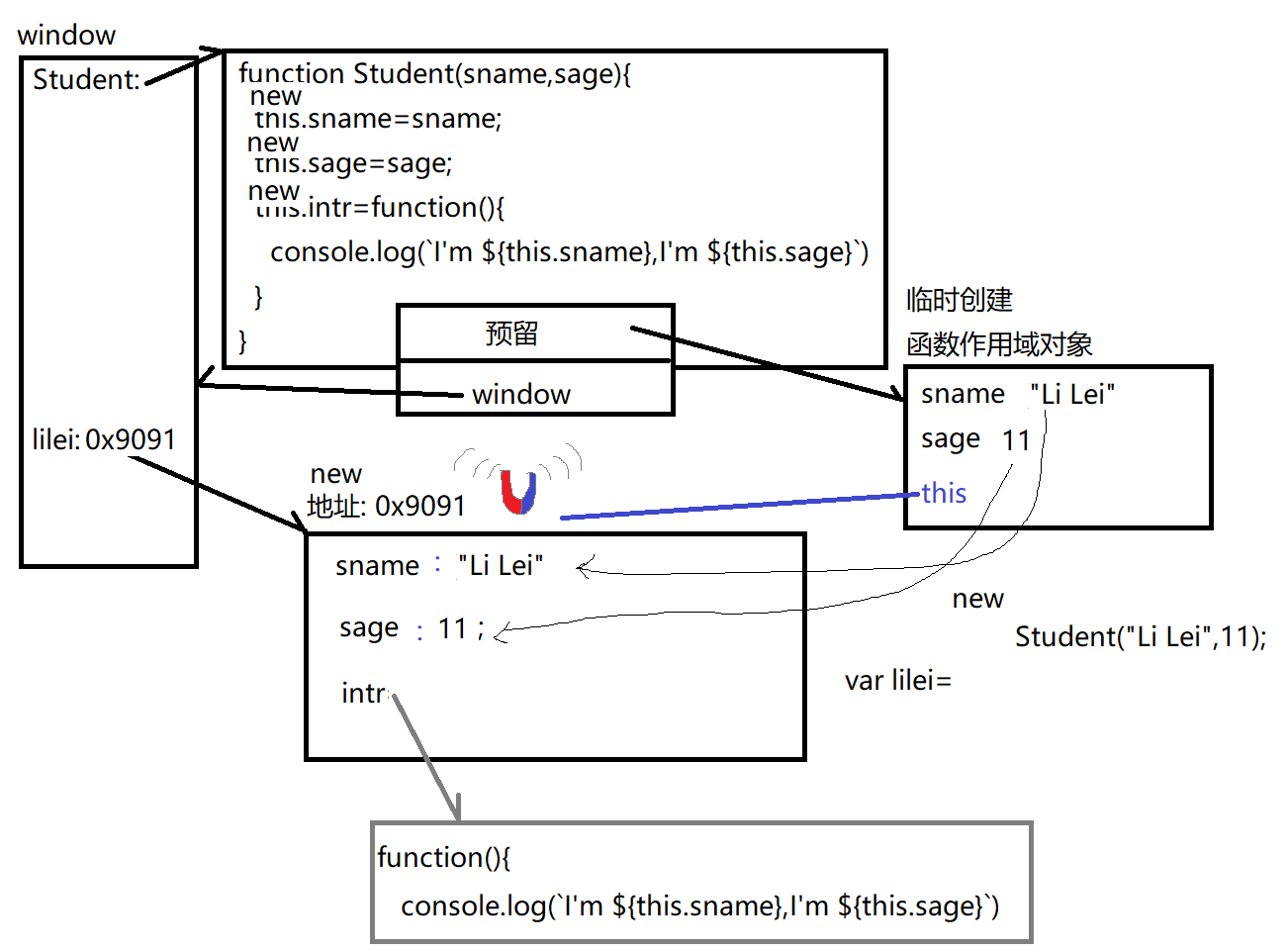
2). 让新创建的子对象自动继承构造函数的原型对象

3). 用新对象调用构造函数:

i. 将构造函数中的所有this，临时指向当前正在创建的新对象

ii. 通过强行赋值的方式，将构造函数内规定的所有属性和方法，强行添加给new刚创建的这个新对象

4). 返回本次创建的对象的地址，保存到等号左边的变量中



5. 如何访问对象中的成员: 通过.操作符访问

(1). 访问保存在一个对象中的属性值: 对象名.属性名

(2). 调用保存在一个对象中的方法: 对象名.方法名()

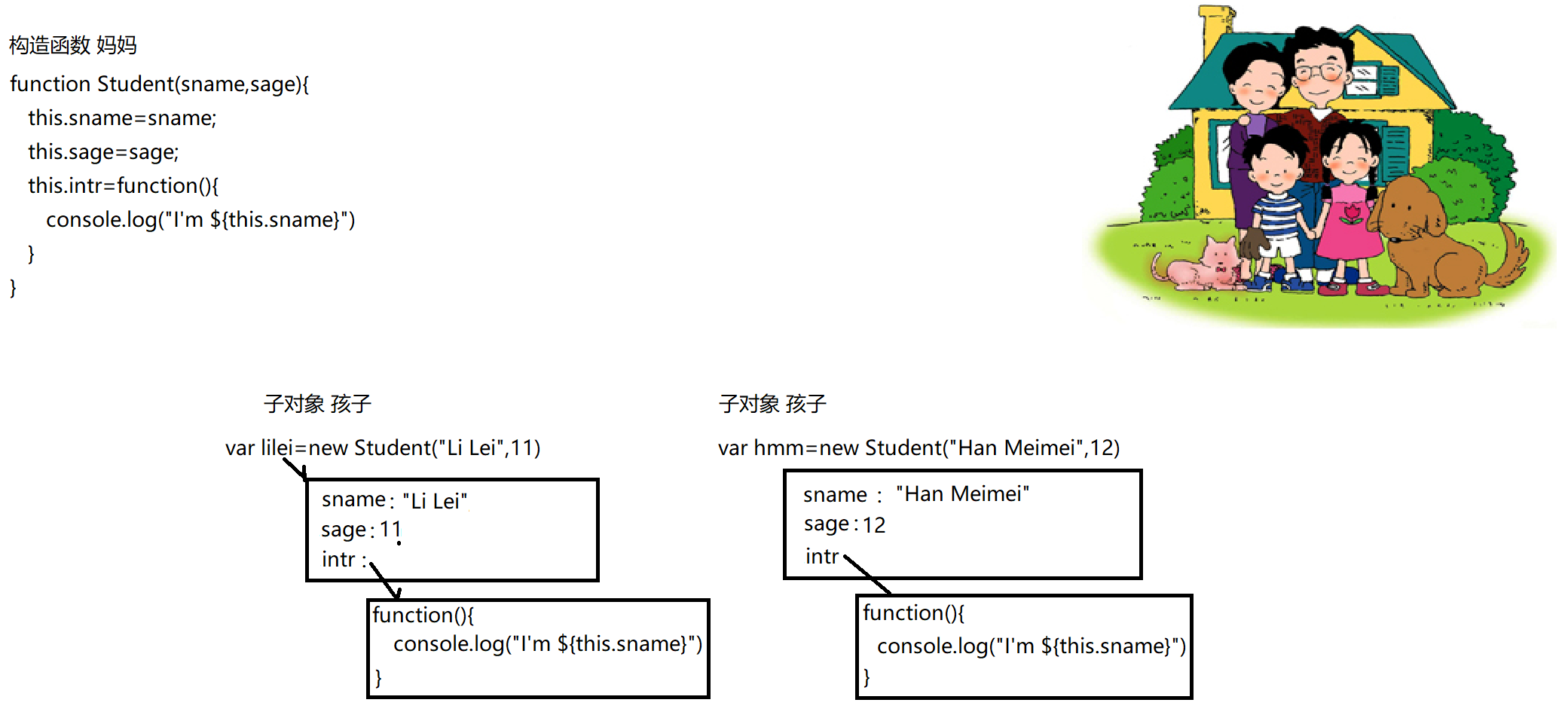
三. 继承:

1. 问题: 构造函数只能重用代码结构，但是浪费内存。

(1). 因为凡是放在构造函数中的方法定义，每创建一个新对象，都要重复创建这个方法对象的副本！——浪费内存！

(2). 所以: 今后构造函数中不应该包含方法的定义！构造函数中应该只包含属性结构的定义。

(3). 所有对象都要使用的公共的方法定义在哪儿呢？——继承



2. 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

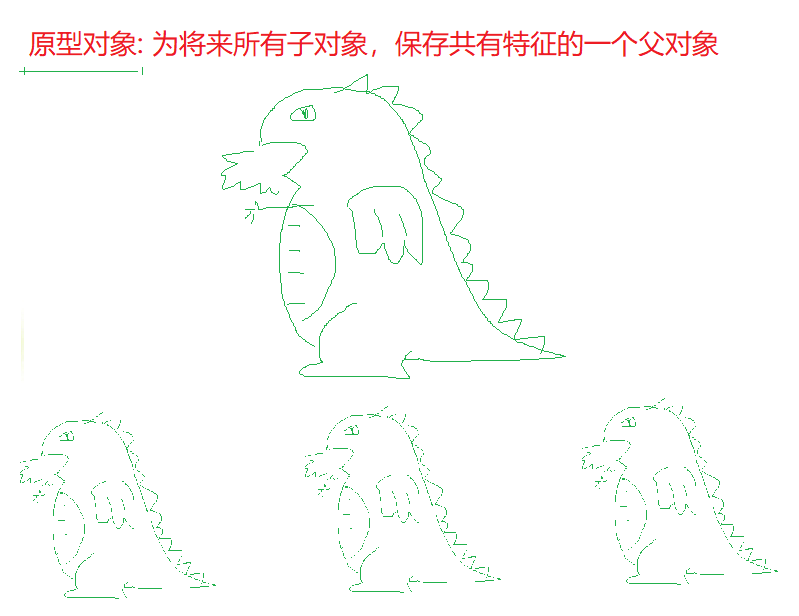
3. 为什么: 既可重用代码，又可节约内存！

4. 何时: 只要同一类型多个对象都需要公共的方法定义时，就要用继承实现

5. 如何:

(1). 找到所有子对象共同的父对象

a. 其实定义每个构造函数时都会附赠一个父对象——原型对象(prototype)



b. 可通过"构造函数.prototype"属性访问到这个暂时为空的原型对象

c. new的第二步: 让新创建的子对象自动继承构造函数的原型对象

new会自动设置 新子对象.\_ \_proto\_ \_=构造函数.prototype

孩子 的 爹 是 妈妈 的 老公

(2). 将所有子对象公共的方法定义添加到共同的父对象原型对象中，只能通过强行赋值的方式:

构造函数.prototype.公共的方法=function(){ ... this.属性名 ...}

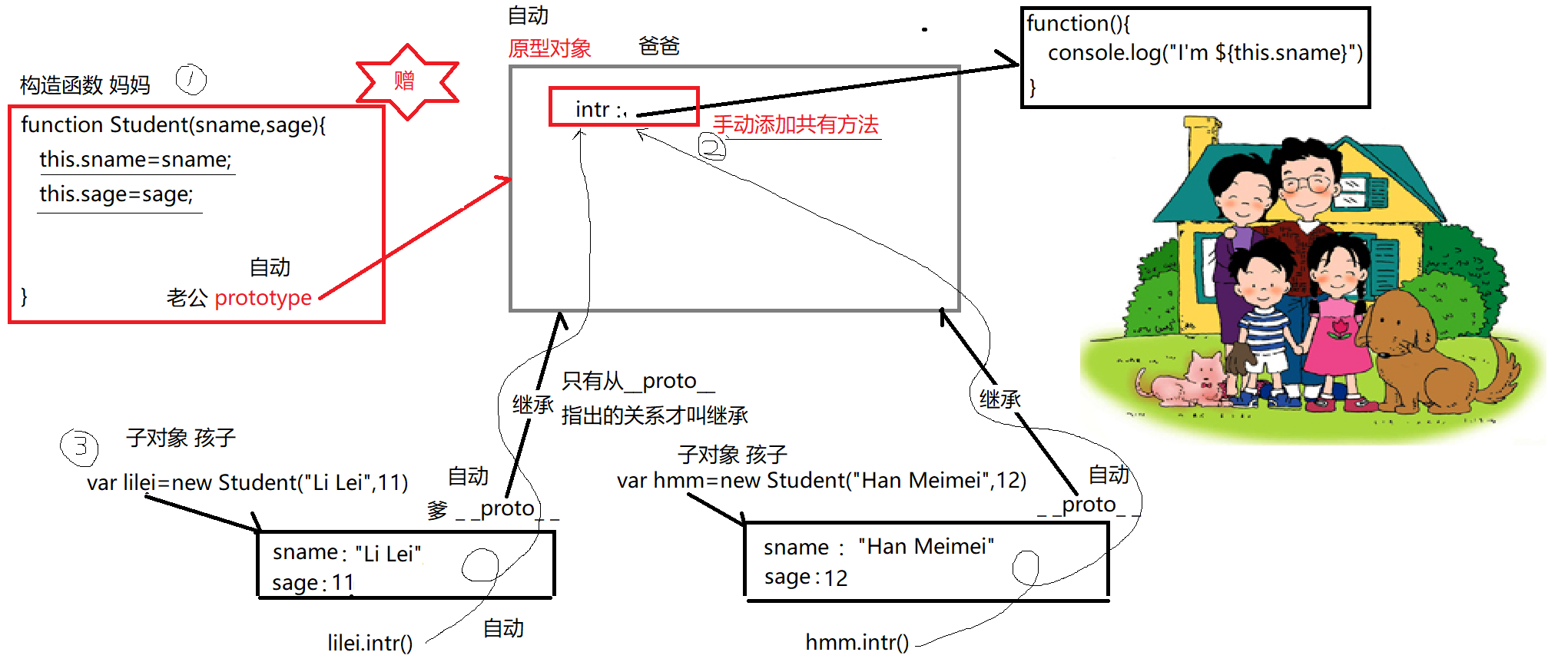
(3). 结果: 所有子对象就可因为继承关系，而直接使用父对象中的公共的方法

a. 当子对象调用原型对象中公共的方法时，js引擎现在子对象本地尝试查找。如果没找到，就会延\_ \_proto\_ \_属性，自动去父对象原型对象中查找！

b. 强调: 判断this~~一定不要看定义在哪儿！~~只看在哪里调用的函数！

调用时，.前是谁！this就临时指谁！

所以，原型对象中公共函数里的this指将来调用这个函数的某一个子对象。



6. 示例: 将所有子对象共有的方法添加到原型对象中

|  |
| --- |
| //构造函数 妈妈  function Student(sname,sage){    this.sname=sname;    this.sage=sage;    /\*从此构造函数中不应该再有方法定义了！\*/  }  console.dir(Student);  //输出定义构造函数时附赠的原型对象爸爸  console.log(Student.prototype);  //想向所有孩子共同的父对象原型对象中添加一个公共的intr函数，让所有孩子公共  Student.prototype.intr=function(){    console.log(`I'm ${this.sname},I'm ${this.sage}`)  }  //两个孩子  var lilei=new Student("Li Lei",11);  var hmm=new Student("Han Meimei",12);  console.log(lilei);  console.log(hmm);  //判断this,不要看定义在哪儿！只看在哪儿调用！  lilei.intr();  hmm.intr();  //亲子鉴定:  //lilei的爹是不是Student的老公？  console.log(lilei.\_\_proto\_\_==Student.prototype);//true  //lilei和hmm的爹是不是同一个人？  console.log(lilei.\_\_proto\_\_==hmm.\_\_proto\_\_);//true |

7. 自有属性和共有属性:

(1). 自有属性: 保存在子对象内部，归某个子对象独有的属性

(2). 共有属性: 保存在原型对象中，归多个子对象共有的属性

(3). 获取属性值时: 两种属性都可用"子对象.属性名"方式访问

(4). 修改属性值时:

a. 自有属性: 可以用"子对象.属性名=新值"方式修改

b. 共有属性: 必须用"原型对象.属性名=新值"方式修改

~~不能用"子对象.属性名=新值"方式修改~~

如果硬要用子对象修改共有属性的值，结果会为这个子对象隐式添加一个自有的同名属性。从此，在这一个属性的使用上，这个子对象和其他子对象，分道扬镳！再无关系！

(5). 示例: 比较自有属性和共有属性:

|  |
| --- |
| //构造函数 妈妈  function Student(sname,sage){    this.sname=sname;    this.sage=sage;  }  //为两个孩子添加共有属性className  Student.prototype.className="初一2班"; //共有  //两个孩子  var lilei=new Student("Li Lei",11);  var hmm=new Student("Han Meimei",12);  var lily=new Student("Lily",10);  console.log(lilei);  console.log(hmm);  console.log(lily);  //输出三个孩子的班级  console.log(lilei.className,hmm.className,lily.className);  //输出三个孩子的自有属性sname  console.log(lilei.sname,hmm.sname,lily.sname);  //过了一年，三个孩子都升了一级  //正确:  //Student.prototype.className="初二2班";  //输出三个孩子的班级  //console.log(lilei.className,hmm.className,lily.className);  //错误:  lilei.className="六年级2班";  //输出三个孩子的班级  console.log(lilei.className,hmm.className,lily.className);  //后果:  //过了一年，孩子们应该都升一级  Student.prototype.className="初二2班";  //输出三个孩子的班级  console.log(lilei.className,hmm.className,lily.className); |

8. 利用原型对象为某种类型的所有子对象添加新共有方法:

(1). 问题: 如果项目中经常对一类对象执行某一种操作，但是该类型原型对象中，又没有提供这个常用的函数，应该怎么办？

(2). 解决: 为这个类型的原型对象，添加一个自定义的共有函数。

(3). 比如: 经常需要对数组中所有元素求和！

(4). 如何自己写:

a. 其实ES标准中已经有11种内置类型/对象:

String Number Boolean ——包装类型(看小程序中视频)

Array Date RegExp Math(不能new)

Error

Function Object

global(全局作用域对象，在浏览器中被window代替，不能new)

b. 除Math和global之外的9种内置类型都包含构造函数和原型对象, 比如:

1). Array其实是一个function Array(){ ... }构造函数

所以创建数组时可以: var arr=new Array()

其实[] 是 new Array()的简写

2). Array也一定有一个附赠的原型对象: Array.prototype

其中保存着所有数组共用的方法: .sort() .reverse() .push() .slice() ... ...

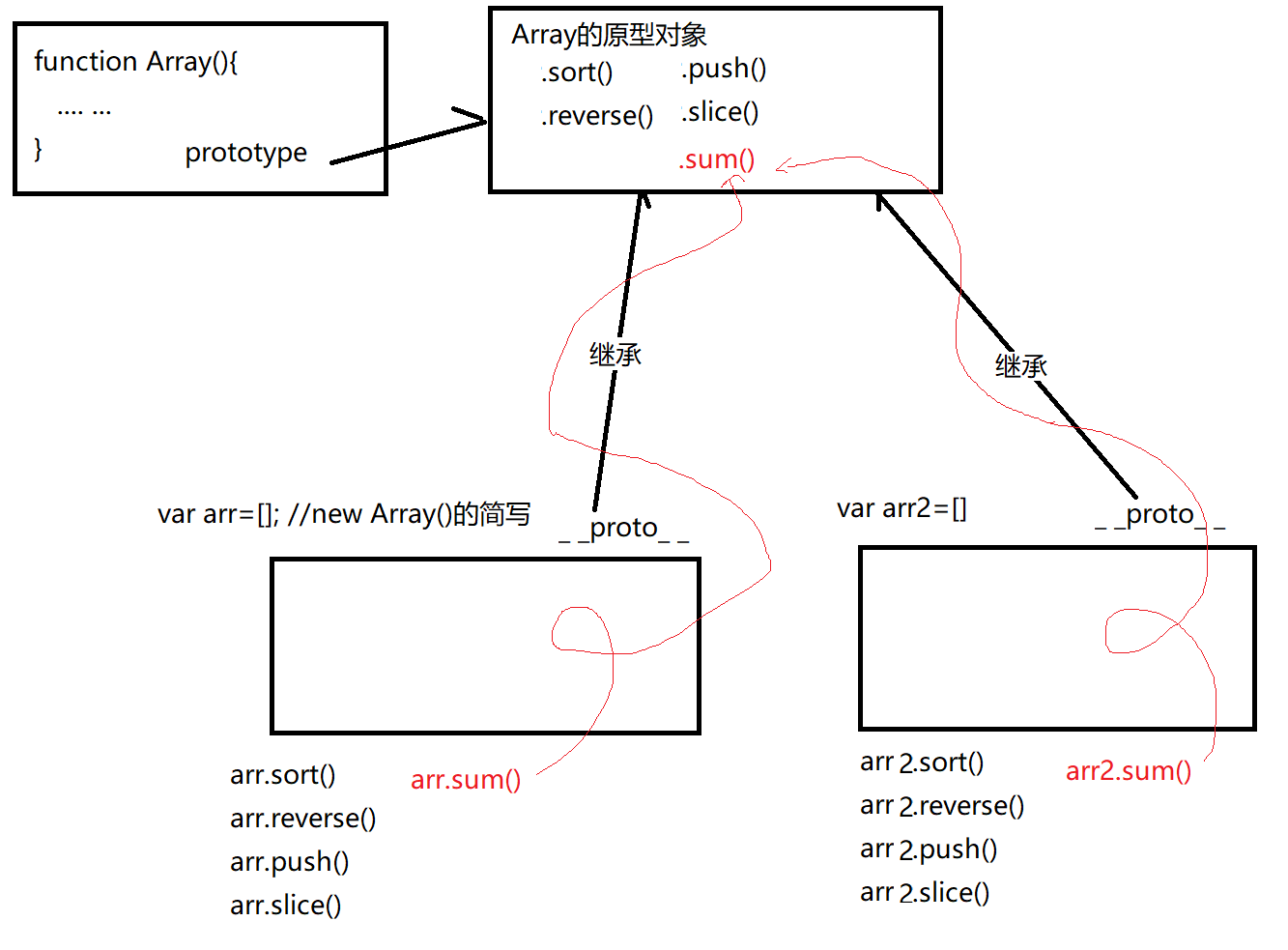
c. 如果数组家缺一个求和的函数sum()，就自己定义一个sum()函数放到数组的原型对象中

Array.prototype.sum=function(){

... ...

}

d. 结果: 所有数组家子对象都可享用这个原型对象中的sum函数了



(5). 示例: 为数组家添加一个共用的sum函数可对数组中元素求和

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <script>      //向数组的原型对象中添加一个sum函数      Array.prototype.sum=function(){        console.log(`调用我们自己在Array.prototype中定义的sum函数`);        var total=0;        for(var i=0;i<this.length;i++){          total+=this[i]        }        return total;      }    </script>  </head>  <body>  <script>  //希望任意一个数组调用sum()函数，就可求和；  var arr1=[1,2,3];  console.log(arr1.sum());//6  var arr2=[1,2,3,4,5];  console.log(arr2.sum());//15  </script>  </body> |

(6). 其实今后想了解一个类型中还有哪些好用的函数辅助我们开发，都可查看该类型的原型对象:

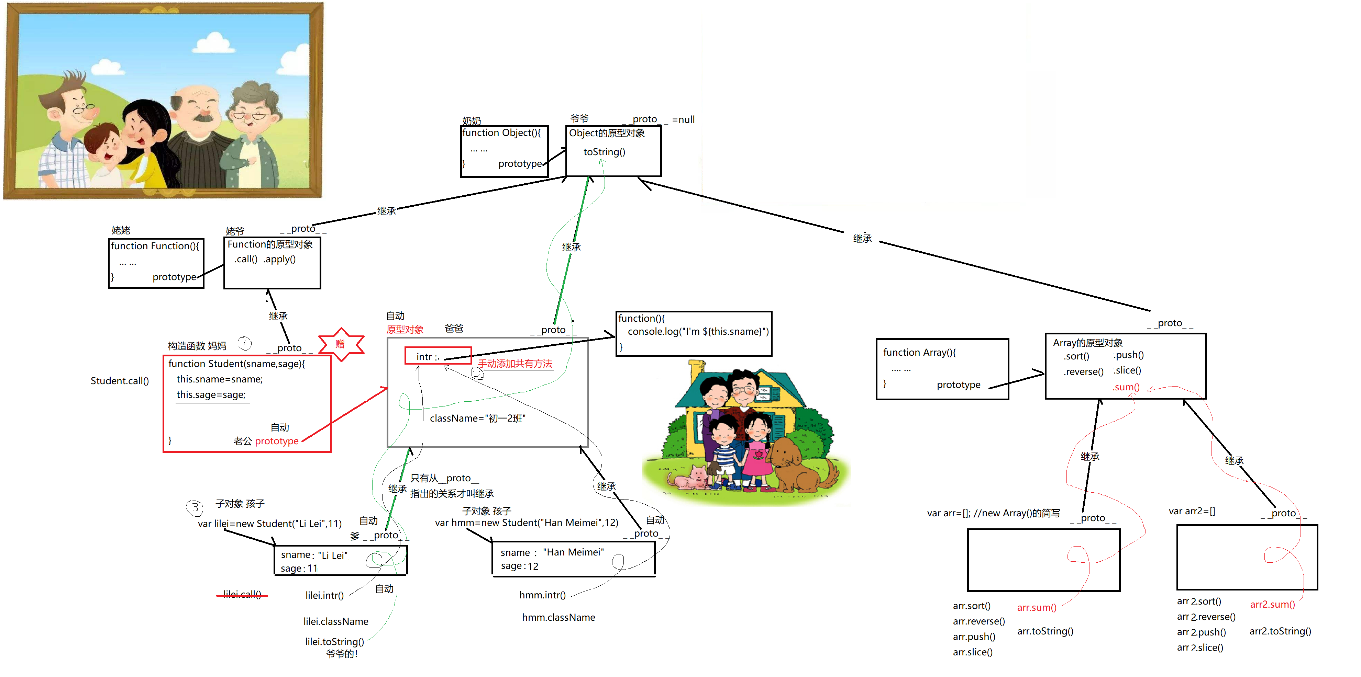
比如: String.prototype、Array.prototype等

9. 原型链:

(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). 保存着一个对象可用的所有属性，包括自有属性和共有属性

(3). 控制着属性的使用顺序: 先自有,再共有,然后再延原型链向上找！



总结: 目前为止this的用法: 4种:

1. obj.fun() fun()中的this->.前的obj对象

2. new Fun() Fun()中的this->new创建的新对象

3. fun() 或 (function(){ ... })() 中的this->window

4. Student.prototype.intr=function(){ ... this.sname ...} 中this->将来调用这个函数的某个子对象，比如lilei或hmm。

总结: 面向对象:

1. 封装:

(1). 只创建一个对象: {}

(2). 反复创建多个相同结构的对象: 构造函数

2. 继承:

(1). 今后所有子对象共有的方法和属性值，不要定义在构造函数中，而应该添加到原型对象中: 构造函数.prototype.共有方法=function(){ ...this.属性名... }

作业:

1. (必须)看小程序视频学习多态:

小程序->在线->JSCORE->day05: 3. 多态 override toString()

2. (必须)看小程序视频学习对象相关高频笔试题：

小程序->在线->JSCORE->day04:

高频笔试题 深克隆

高频笔试题 去掉数组中重复的元素

高频笔试题 查询字符串search转对象

高频笔试题 统计字符串中每种字符出现的次数

3. (必须)看小程序视频学习继承相关高频笔试题：

小程序->在线->JSCORE->day04: 判断一个对象是不是数组类型，共有几种方式

4. (必须)看小程序视频学习包装类型:

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

5. (扩展)看小程序视频学习面向对象相关高频笔试题:

小程序->在线->JSCORE->day04: 所有以"高频笔试题" 开头的小视频