正课：

1. 面向对象

2. ES5

1. 面向对象

1. 封装: 3种方式

1. 如果定义对象时就知道对象的成员内容:

var obj={ ... }

2. 如果定义对象时，暂时不知道对象的成员内容:

var obj=new Object() 或 {}

然后，随后的任何时候，都可以通过强行赋值的方式为对象添加新属性。

3. 如果反复创建多个相同结构的对象时，都用构造函数

2步:

1. 定义构造函数:

function 类型名(形参列表){

this.属性名=形参;

... = ...

//不要把方法定义放在构造函数中

}——放在构造函数内的成员，都会给每个孩子单独创建一个副本。如果希望多个孩子共用的成员，应该定义在原型对象中。而不是构造函数里。

2. 用new 调用构造函数，创建一个新对象

var obj=new 类型名(属性值列表)

new做了四件事:

1. 创建一个新的空对象

2. 让新对象继承构造函数的原型对象

3. 执行构造函数的内容，new自动将所有的this吸引到新对象上。操作this，等效于操作新对象。通过强行赋值的方式给新对象添加规定的属性。

4. 返回新对象的地址保存在变量中

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: 节约内存，代码重用！

何时: 今后只要多个子对象共用的属性值和方法，都要定义在父对象中一份即可。

如何:

通过强行赋值的方式，将共有的属性值和方法，添加到构造函数的原型对象中:

类型名.prototype.共有成员=值

结果: 孩子.共有成员

共有属性和自有属性:

自有属性: 保存在对象本地，只归对象自己所有的属性

共有属性: 保存在父对象中，归多个对象共有的属性

获取属性值: 两者没有差别: 子对象.属性

修改属性值:

自有属性: 只能用子对象修改

共有属性: 虽然可以用子对象读取，但是绝不能用子对象修改。只能用父对象统一修改。

如果强行用子对象修改共有属性，则:

1. 不会修改原型对象中的共有属性

2. 而是给当前子对象添加同名自有属性

结果，从此，当前子对象与其它子对象以及父对象，在这个共有属性的使用上分道扬镳，再不同步。

原型链:

什么是: 多级父对象组成的链式结构

包含了: 一个对象的所有父级祖先对象

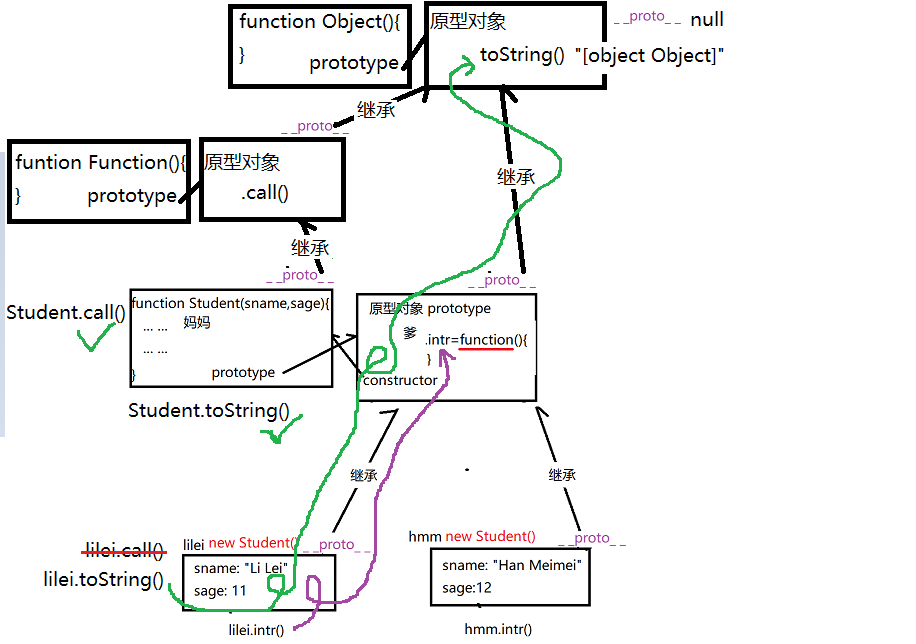
顶级父对象是Object类型(function Object(){}和Object.prototype)

所有对象，最终都会继承到Object.prototype对象上。Object.prototype对象中的成员所有其他对象都可使用。

控制着对象方法的使用顺序:

先在对象本地查找使用

对象本地没有就延原型链向上查找，直到找到为止。



选择: 我要定义一个函数:

1. 只想让lilei一个人用: lilei.fun=function(){}

2. 只想让学生类型下的所有子对象用，不想让别的类型的子对象使用: 放在学生类型的原型对象中:

Student.prototype.fun=function(){ ... }

3. 想让所有对象都能用: 放在顶级父对象中:

Object.prototype.fun=function(){ ... }

内置类型的原型链结构:

内置类型: ES标准中规定的，浏览器已经实现的类型。

包括: 11种:

Number String Boolean

Array Date RegExp Math

Error

Function Object

global (浏览器中其实就是window)

其实内置类型(除Math和global之外)都是一个家庭(类型)，都至少由两部分组成:

构造函数和原型对象:

构造函数负责创建该类型的子对象——妈妈

原型对象负责保存所有孩子共用的方法——爸爸

比如: 我的项目中要求所有的数组都能调用一个函数sum()，可对数组中的所有元素自动求和:

var arr1=[1,2,3]

arr1.sum() => 6

var arr2=[1,2,3,4,5]

arr2.sum() => 15

只要在数组类型的原型对象中，添加sum()函数

Array.prototype.sum=function(){ ... }

3. 多态:

什么是: 一个函数不同情况下表现出不同的状态

包括2种情况:

1. 重载:

2. 重写（override）:

什么是: 在子对象中定义一个和父对象中同名的成员

为什么: 不是所有继承来的都是好用的。

何时: 如果觉得父对象继承来的成员不好用，可以自己定义同名成员覆盖父对象中。

因为使用成员时，都是现在自己本地找。如果自己本地有，就不去父对象中查找。

如何: 只要在子对象中定义和父对象中同名的成员即可。

修改继承关系: 2种:

1. 仅修改一个对象的爹:

子对象.\_\_proto\_\_=新的父对象;

||

Object.setPrototypeOf(子对象, 新的父对象)

设置 原型 的

设置子对象的原型为新的父对象

2. 同时修改所有子对象的爹:

其实就是修改构造函数的原型对象:

构造函数.prototype=新father

时机: 必须在没有创建子对象之前换掉

总结: 封装，继承，多态

1. 封装: 2种:

1. 用{}创建一个对象

2. 用构造函数反复创建多个对象

2. 继承: 所有方法都定义在原型对象中一份即可

如果觉得父对象不满意，可以换爹

3. 多态: 如果觉得父对象继承来的成员不好用，可以在本地重写同名的成员。

2. ES5:

什么是: ECMAScript的第5个升级版本

javascript=ECMAScript + DOM + BOM

核心语法 操作网页 操作浏览器窗口

为什么: 因为js中有很多广受诟病的缺陷

包括:

1. 严格模式:

什么是: 比普通js运行机制，要求更严格的模式

为什么: 因为js中有很多广受诟病的缺陷

何时: 今后所有js程序必须运行在严格模式下

如何: 在当前作用域的顶部，添加: "use strict";

这句话之后的程序都运行在严格模式下！

比普通js模式更严格的规定包括:

1. 禁止给未声明的变量赋值:

普通js中: 给一个未声明的变量强行赋值，会自动在全局创建该变量。——造成内存泄漏

严格模式下: 只要给一个未声明的变量赋值，就会报错！——好处: 避免内存泄漏

总结: 今后要使用一个变量，必须先声明。

只要未声明的变量，就不能用！

2. 静默失败升级为错误:

静默失败: 执行不成功，还不报错！——缺点不知道原因，无法调试！

严格模式下: 所有静默失败都升级为错误

3. 普通函数调用或匿名函数自调中的this不再指向window，而是值为undefined

后果，如果误给this赋值，会报错。因为undefined后，什么也不能加。只要加，就报错！——好事儿，避免了无意中生成全局变量，造成内存泄漏和全局污染。

4. 禁止使用arguments.callee

arguments.callee: 专门在函数执行时，在函数内部自动引用当前函数自己的特殊变量。

何时: 递归调用

递归的问题一: 如果在函数内写死函数名，则万一函数名变化，忘记修改内部，立刻程序就出错！——紧耦合

解决: arguments.callee 代替写死的函数名——松耦合: 修改函数名，函数内可不用修改。

递归的问题二: 递归的效率极低——重复计算量太大。

所以严格模式禁用了arguments.callee，等效于暗示你不要使用递归算法！

解决: 几乎所有的递归都可以用循环代替——难度: 用海量数据找规律

总结: 今后，尽量用循环代替递归算法。

单词列表:

1. override 遮挡/覆盖 重写

2. define 定义

3. property 属性

4. writable write写 able可以xxx writable 可修改的

5. assign 赋值 =