正课:

\*\*\*OOP

1. 封装

2. 继承

3. 多态

4. 自定义继承

1. 封装: 3种:

1. 直接量:

var obj={

属性名:值,

... : ... ,

方法名(){

... this.属性名 ...

}

}

问题: 对象自己的方法，要使用自己的属性值，缺找不到！

原因: 不加.访问的变量，只能在作用域链中查找，无法自动进入对象中查找

解决: 错误: 在属性前加"对象."

在对象内，禁止写死对象名

因为对象名其实仅是一个普通变量名，很有可能发生变化

正确: 用this.属性名:

this: 在函数执行时，自动创建的一个关键词

自动指向正在调用当前函数的.前的对象

为什么: 不受对象名的影响

总结: 今后，只要对象自己的方法，想访问自己的属性，必须用this.属性名

2. 用new: 2步:

1. 先创建一个空对象

var obj=new Object(); //new可省略,()也可省略,但不能同时省

2. 向空对象中添加新成员:

obj.属性名=值;

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

说明: js中的对象，可随时，通过强行赋值的方式，添加新成员

何时: 如果在创建对象时，暂时不知道对象的成员

揭示: js中的对象，底层其实就是关联数组

相同: 1. 都可用.或[]随时添加/访问成员

2. 访问不存在的成员，不报错，返回undefined

3. 都可用for in遍历每个成员

不同: 对象比关联数组用法简单！

今后，都是用对象来代替关联数组使用！

问题: 一次只能创建一个对象

如果反复创建相同结构的多个对象，重复代码会很多

解决:构造函数

3. 用构造函数(constructor):

什么是: 描述同一类型的多个对象，相同成员结构的函数

第二个作用: 将一个空对象构造为拥有属性和功能的完整对象

何时: 只要反复创建同一类型的多个对象时，都要先用构造函数描述统一的结构，再用构造函数创建对象

如何: 2步

1. 定义构造函数:

function 类型名(属性参数,....){

this.属性名=属性参数;

... = ... ;

this.方法名=function(){

... this.属性名 ...

}

}

2. 使用构造函数反复创建对象:

var obj= new类型名(属性值,...);

调用时的参数值，应和定义构造函数时的属性参数保持一致。

new: 4件事:

1. 创建一个新的空对象

2. 自动让新的子对象继承构造函数的原型对象

3. 调用构造函数，将构造函数中的this执行正在创建的新对象。向新的空对象中强行添加新成员

4. 将新对象地址返回给变量保存

优: 重用结构定义

缺: 浪费内存

2. 继承:

什么是: 父对象的成员，子对象无需创建即可直接使用！

为什么: 代码重用！节约内存！

何时: 所有子对象都拥有相同的属性值和方法定义时，都要用继承来实现

如何: js中所有继承，都是继承原型对象

什么是原型对象: 集中存储所有子对象共有成员的父对象

为什么: 实现继承

何时: 只要实现继承，都要继承原型对象

如何:

创建: 不用手动创建，买一赠一

其实创建构造函数同时，都附赠一个空的原型对象

继承: 不用手动设置

用new创建新的子对象时，会自动设置新对象继承构造函数的原型对象

添加共有成员:

构造函数.prototype.成员=值;

总结: 只要所有子对象共用的成员，都必须集中存储在原型对象中

自有属性和共有属性:

自有属性: 直接保存在对象本地的属性

共有属性: 保存在原型对象中，所有子对象共有的属性

读取: 两者完全一样: 对象.属性名

修改: 自有属性: 只能用子对象改: 子对象.自有属性名=值

共有属性: 只能用原型对象修改:

构造函数.prototype.共有属性名=值

原型链: prototype chain

什么是: 由多级父对象，逐级继承，形成的链式结构

为什么: 为了更高级,更大范围的重用

如何:

所有对象，都有\_\_proto\_\_属性

原型对象的\_\_proto\_\_指向更上级的父对象

所有对象最终都继承自Object.prototype——顶级父对象

内置对象的原型链:

其实每种内置类型都有对应的构造函数和原型对象，也最终都继承自Object.prototype

其中: 内置类型的构造函数负责创建该类型的子对象

内置类型的原型对象负责保存该类型所有子对象共有的API

问题: 旧浏览器不支持新的API

解决: 向旧浏览器中的原型对象中手动添加一个函数

鄙视: 判断一个对象是不是数组类型，有几种方法:

typeof不行！

1. 判断原型对象:

Object.getPrototypeOf(obj)==Array.prototype

判断obj是数组类型的子对象

问题: \_\_proto\_\_是内部属性，本不应该被访问到

解决: 用Object.getPrototypeOf(obj) 代替\_\_proto\_\_

2. 判断构造函数:

实例

obj instanceof Array

判断obj是不是被构造函数Array创造出来的

instanceof 不仅判断直接父类型，而是所有在原型链上的类型，都返回true！

3. 判断对象的内部class属性

每个对象内部，都有一个隐藏的class属性，记录该对象创建时的数据类型

class属性不会随继承关系的改变而改变

问题1: class是内部属性

解决: 只有最顶层的toString()才能输出对象的class属性值

[object class名]

问题2: 内置类型的原型对象中几乎都重写了新的toString()

解决: 用call强行调用:

call: 让一个对象，调用一个本来无法调用到的函数

何时: 只要希望调用一个本无法调用到的函数

如何: 要调用的函数.call(对象)

Object.prototype.toString.call(obj)=="[object Array]"

说明obj的内部属性class的值为"Array"

3. 多态:

什么是: 同一个函数在不同情况下表现出不同的状态

重写: 如果子对象觉得父对象的成员不好用，可在本地定义同名成员，覆盖父对象中继承来的成员

为什么: 体现子对象和父对象之间的差异

何时: 只要子对象觉得父对象的成员不好用，就可以重写！