正课:

\*\*\*\*OOP

什么是

封装

继承

多态

什么是面向对象: 程序中都是用对象结构来描述现实中一个具体事务的属性和功能

为什么: 便于维护

何时: 今后几乎所有大型项目都采用面向对象的思想

如何: 三大特点: 封装，继承，多态

封装: 用一个对象结构集中保存现实中一个事物的多个属性和功能。

什么是对象: 封装一个事物的属性，并提供操作事物的功能的数据结构。

为什么: 现实中任何一个数据或一项功能都属于一个具体事物。——便于维护

何时: 今后只要使用面向对象的方式，首先都要创建对象。

如何: 3种:

1. 对象直接量: var obj={

属性名:值,

... : ... ,

方法:function(){

...this.属性名...

}

... : ...

}

属性，其实就是保存在对象中的一个普通变量

方法，其实就是保存在对象中的一个普通函数

问题: 对象自己的方法，通常要使用对象自己的属性值

错误: 直接使用属性名当变量

正确: this: 引用正在调用函数的.前的对象的关键词

今后，只要对象自己的方法要使用对象自己的属性，必须用this.属性名

访问对象的成员: 成员=属性+方法

访问属性: 对象.属性名 用法和普通变量完全一样

特殊: 要访问的属性不存在，不报错！

返回undefined

调用方法: 对象.方法名() 用法和普通函数完全一样！

2. 用new: 2步:

var obj=new Object(); //先创建一个空对象

obj.属性名=属性值;//再向空对象中添加属性和方法

obj.方法名=function(){

...this.属性名...

}

其实js中的对象可在任何时候添加新属性和方法

特殊: 要赋值的属性不存在，不报错!

而是在对象中动态添加该属性

js中对象的本质，就是关联数组。

其实访问成员: 对象["属性名"]

对象["方法名"]()

. vs [] : 如果要访问的属性名是固定的，就用.访问

如果要访问的属性名是动态生成的，就用[]

问题: 只能创建一个单独的对象

反复创建多个相同结构的对象时，代码重复严重

解决: 构造函数

3. 用构造函数反复创建多个相同结构的对象

什么是构造函数: 描述同一类型的所有对象共同成员结构的函数。

为什么: 代码重用！

何时: 只要反复创建同一类型的多个对象时

如何: 2步:

1. 定义构造函数:

function 类型名(属性参数列表){

this.属性名=参数;

... = ... ;

this.方法名=function(){

...this.属性名...

}

}

2. 调用构造函数创建一个对象:

var obj=new 类型名(属性值列表)

原理: new: 4件事:

1. 创建新的空对象，将构造函数中的this->新对象

2. 让新对象自动继承构造函数的原型对象

3. 执行构造函数中的语句，向新对象中添加新成员并赋值

4. 返回新对象地址保存到变量中

继承: 父对象的成员，子对象无需重复创建，就可直接使用。

为什么: 代码重用，节约内存

如何:

原型对象: 集中保存同一类型的子对象共有成员的父对象

创建: 在定义构造函数时，自动创建(买一赠一)

如何:

1. 使用构造函数创建子对象时，会自动设置子对象继承构造函数的原型对象

2. 放在原型对象中的成员，所有子对象共用

如何访问构造函数的原型对象:

构造函数.prototype.成员名=值/function(){...}

总结: 今后，只要希望所有子对象共有的成员，都要添加到构造函数的原型对象中。而不应该放在构造函数内。

共有属性和自有属性:

共有属性: 添加到原型对象中，所有子对象共有的属性

自有属性: 直接保存在对象本地的属性

访问: 读取: 两者用法完全相同: 对象.属性

先在对象本地找，没有，才去父对象中找

修改: 自有属性: 对象.属性=值

共有属性: 子对象不允许修改共有属性

必须通过构造函数

构造函数.prototype.属性=值

判断属性是否可用/包含:

"成员" in 对象

返回bool

强调: in不仅判断自有属性，而且判断父对象中的共有属性。

内置对象的原型对象:

内置对象，也有构造函数和原型对象

构造函数负责创建指定类型的对象

原型对象负责保存该类型子对象共有的API

解决新旧浏览器API兼容性问题: 2步:

1. 判断当前浏览器是否支持API

2. 如果不支持，就自定义一个API放入该类型的原型对象中

原型链: 由多级父对象逐级继承形成的链式结构

保存了所有对象的成员(属性和方法)

定义了成员的使用顺序:

先用自有成员, 自己没有，才延原型链向父对象查找

vs 作用域链:

保存了所有变量(全局/局部)

定义了变量的使用顺序: 先用局部，没有，才延作用域链先父级作用域查找

总结: 只要不需要对象. 就可直接访问的变量都存在作用域链。

只要必须用对象.才能访问的对象成员都存在原型链中。

多态: 同一个方法，在不同情况下表现出不同的状态。

js中仅支持重写(override)

什么是重写: 如果子对象觉得父对象的成员不好用，可自定义自有成员，覆盖父对象中的成员。

为什么: 从父对象继承来的成员，不一定总是想要的

何时: 只要子对象觉得父对象的成员不好用

自定义继承:

1. 仅修改两个对象间的继承关系

Object.setPrototypeOf(child, father)

让child继承father( 设置child的\_\_proto\_\_继承father)

问题: 一次只能修改一个对象的继承关系

解决:

2. 修改构造函数的原型对象来修改所有子对象的父对象

构造函数.prototype=新father

强调: 必须在创建子对象之前更换

3. 两种类型间的继承:

问题: 如果两种类型间有部分相同的属性结构和方法

优化: 3步:

1. 定义抽象父类型:

父类型构造函数中定义公共的属性结构

父类型原型对象中定义公共的方法

2. 在子类型构造函数中借用父类型构造函数

错误: 直接调用父类型构造函数()

因为如果一个函数没有用new或任何对象.调用，则this默认->window

正确: 用call/apply，将父类型构造中的this临时替换为子类型构造函数中的this

父类型构造.call(this,参数....)

3. 设置子类型的原型对象继承父类型的原型对象