正课:

1. \*\*\*\*\*面向对象OOP

三大特点: 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物。

什么是对象: 程序中封装现实中一个事物属性和功能的存储空间。

为什么: 现实中，任何数据都有明确的归属，都不是孤立的。

何时: 只要用程序描述现实中一个事物，都要将事物的属性和功能封装在一个对象中

如何: \*\*\*\*三大特点: 封装，继承，多态

封装:

什么是封装: 将现实中一个事物的属性和功能集中定义在一个对象结构中:

事物的属性会成为对象的属性

事物的功能会成为对象的方法

为什么封装: 让每个数据都有其专门的归属，便于维护和操作

何时封装: 只要使用面向对象，都要先将数据封装在对象中，再使用。

如何封装: 3种:

1. 对象直接量:

var obj={//创建一个新对象

属性名:值,

... : ... ,

//方法名:function(参数列表){

方法名(参数列表){//ES6

this.属性名...

},

方法名(...){...}

}

何时使用直接量: 如果创建对象时就已经知道对象的成员。

\*\*\*\*\*问题: 对象自己的方法，要访问自己的属性:

错误: 对象名.属性名, 一旦对象名修改，方法内的对象名要同时修改。不便于维护

正确: 在方法内使用关键词this自动指代当前对象本身。

this.属性名

优: 即使对象名修改，this也可自动获得当前对象本身，和对象名无关

this可翻译为: 当前对象的/自己的

总结: 今后只要对象自己的方法中，要使用自己的属性，就必须用this.属性名

2. 用new: 2步:

1. 创建空对象: var obj=new Object();

2. 向空对象中添加新属性:

obj.属性名=值;

obj.方法名=function(){ this.属性名 }

何时使用: 如果创建对象时暂时不知道对象的成员

对象创建后，随时可添加新属性。

js中对象本质: 其实就是一个关联数组

对象其实是关联数组的简化版用法:

关联数组 对象

访问元素: ym["属性名"] ym.属性名

创建: 2步: 先创建空[] 可用直接量{}一次性创建

再添加新元素

问题: 一次只能创建一个对象

反复创建多个相同结构的对象，代码冗余太多，不便于维护

解决:

3. 用构造函数反复创建多个相同结构的对象:

构造函数: 专门描述一类对象统一结构的函数

还用于将一个新的空对象装修成想要的结构并存入数据。

何时: 反复创建多个相同结构的对象

都要先用构造函数描述统一的结构

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=属性参数;

... ...

this.方法名=function(){

this.属性名

}

}

2. 用new调用构造函数创建新对象:

var obj=new 类型名(属性值列表);

new 4件事:

1. 创建一个新的空对象

2. 自动设置新对象继承构造函数的原型对象

3. 调用构造函数，向新对象中添加新属性

4. 返回新对象的地址保存在变量中

如何访问对象的成员:

成员=属性+方法

访问属性: 对象.属性名 用法和普通的变量完全一样

属性其实就是保存在对象内的变量

调用方法: 对象.方法名() 用法和普通的函数完全一样

方法其实就是保存在对象内的函数

.读作"的"

其实，也可用对象["属性名"]方式访问

问题: 方法定义在构造函数内。每创建一个新对象，都会重复创建相同的函数副本——浪费内存！

总结: 构造函数: 优: 代码重用！ 缺: 无法节约内存

解决:

\*\*\*\*\*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: 不但可代码重用，且还可节约内存

何时: 只要多个子对象，拥有相同的成员时，都要将相同的成员，仅保存在父对象中一份即可。所有子对象共用。

如何:

原型对象(prototype): 专门集中保存同一类型的多个子对象，共有成员的父对象。

如何获得原型对象:

1. 买一赠一: 创建构造函数同时，已经自动创建了该类型的原型对象。构造函数的prototype属性引用着原型对象。原型对象的constructor也引用着构造函数对象

2. 自动继承: 创建子对象时，会自动设置新对象的

\_ \_proto\_ \_属性继承构造函数的原型对象

如何向原型对象中添加共有属性:

构造函数.prototype.属性/方法名=值/function(...){...}

强调: 原型对象中的方法，要想访问对象自己的属性，也必须加this.

总结:

每个子对象，值不同的属性，都要定义在构造函数中

所有子对象共有的相同方法和属性值，都要集中定义在原型对象中。

共有属性和自有属性:

自有属性: 直接保存在当前对象本地的属性

共有属性: 保存在原型对象中，所有子对象共用的属性

相同: 取值时, 对象.属性

不同: 修改时:

自有属性可直接通过自对象修改:

子对象.自有属性=值

共有属性只能通过构造函数的原型对象

构造函数.prototype.共有属性=值

原型链: 由各级父对象逐级继承形成的链式结构

任何对象都有\_\_proto\_\_属性指向其父对象

保存了所有对象中的成员

控制着对象成员的访问顺序: 优先使用自有属性

自己没有才延原型链向父级找

只要找到就不再向上找

内置对象的原型对象:

其实每种内置对象也都有一对儿构造函数和原型对象:

其中:

构造函数负责创建新对象:

比如: var arr=new Array();

var now=new Date();

var reg=new RegExp();

特例: Math和window不是构造函数，不能new！

原型对象负责集中存储该类型可用的所有API

比如: arr.sort() arr.push() arr.slice()

因为Array.prototype:{

sort(){...},

push(){...},

slice(){...},

...

}

解决浏览器兼容性问题: 旧浏览器无法使用新API

2步:

1. 判断: 如果当前浏览器的指定类型的原型中不包含想要的API

if(!"indexOf" in Array.prototype)

其中in用于检查左边的成员名是否在右边的对象中或对象的原型链上

if(typeof Array.prototype.indexOf!=="function")

2. 如果没有就向原型中添加一个新函数

Array.prototype.indexOf=function(){

this //代表当前数组对象

}

问题: 从父对象继承来的成员不一定都是想要的

解决:

多态: 同一个函数在不同情况下，表现出不同的状态

重写(override): 如果子对象觉得父对象的成员不好用，就可在子对象本地定义同名成员，覆盖父对象的成员。

为什么: 从父对象继承来的成员不一定都是想要的

何时: 如果子对象觉得父对象的成员不好用

总结: 面向对象三特点:

封装: 将事物的属性和功能集中定义在一个对象中

为什么: 便于维护

继承: 父对象的成员，子对象无需重复创建，可直接使用

为什么: 代码重用，节约内存——偷懒

多态: 如果父对象的成员不好用，可在子对象中重写同名成员。

为什么: 为了体现父子对象间的差异

自定义继承:

何时: 只要希望获取其它对象现成的成员时

如何: 3种:

1. 直接修改一个对象的\_\_proto\_\_属性指向新父对象

child.\_\_proto\_\_=father

问题: \_\_proto\_\_是内部属性，不推荐使用

解决: Object.setPrototypeOf(child,father)

2. 通过修改构造函数的原型对象来批量修改所有子对象的父对象:

构造函数.prototype=father

强调: 时机: 在开始创建子对象之前，就要换。

3. 两种类型间的继承: 更像Java的继承

问题: 如果两种类型间拥有部分相同的属性结构和方法定义。

解决: 抽象出一个父类型:

1. 父类型的构造函数中包含子类型相同的部分属性

2. 父类型的原型对象中包含子类型相同的部分方法

3. 在子类型构造函数中借用父类型构造

错误: 直接调用父类型构造:

任何一个函数不用.不用new调用，其中的this默认指向window

解决: 用call强行调用，并替换this为指定对象:

父类型构造.call(this,参数...)

call: 可强行调用一个函数，并替换函数中的this

何时: 如果函数中默认的this不是想要的

如何: fun.call(obj,参数值...)

调用fun，替换fun中的this为obj

4. 让子类型原型继承父类型原型

Object.setPrototypeOf(

子类型.prototype, 父类型的prototype

)