正课:

1. \*\*\*\*\*OOP:

封装， 继承， 多态

什么是面向对象编程: 程序中都是用对象来描述显示中一个具体事物。

为什么: 便于维护和使用数据

何时: 今后只要维护的数据量或种类多时，都要用对象的方式，集中管理每个事物的属性和功能。

什么是对象: 内存中集中存储一个事物的属性和功能的程序结构

如何: 三大特点: 封装，继承，多态

封装: 创建一个对象

什么是: 将一个事物的属性和功能集中保存在一个对象中，再起一个统一的名字

事物的属性会成为对象中的属性

事物的功能会成为对象中的函数

为什么: 便于维护和查找事物的属性和功能

何时: 只要使用面向对象，都要先将事物的属性和功能集中定义在对象中。

如何: 3种:

1. 只创建一个单独的对象: 2种:

1. 对象直接量:

var obj={//创建一个对象

属性名: 属性值,

... : ... ,

//方法名: function(){...},//旧

方法名(){

...this.属性名...

}, //ES6

}

属性和方法，统称为对象的成员

对象的方法如何访问对象自己的属性:

错误: 直接使用不加.的属性名

因为不加.的属性默认只能在作用域链中找

无法自动进入对象中查找属性

正确: this

什么是: 自动获得正在调用当前函数的.前的对象的关键词

为什么: 不带this的普通变量，无法进入当前对象内查找属性

何时: 只要对象的方法，想访问自己的属性，都必须加this.

如何访问对象的成员:

如何访问对象的属性: 对象.属性名

其实属性就是保存在对象中的变量，用法和普通变量完全一样！

如何调用对象的方法: 对象.方法()

其实方法就是保存在对象中的函数，用法和普通函数完全一样！

2. 用new: 2步:

1. 先创建空对象: var obj=new Object();

2. 向空对象中添加新属性和方法:

obj.属性名=值;

obj.方法名=function(){

...this.属性名...

}

何时: 创建对象时，暂时不知道对象中的成员

强调: js中的对象，可在任何时候，添加任何新属性和方法。

如何: 只要尝试给不存在的属性和方法赋值时，对象会自动创建该属性。

2. 反复创建多个相同结构的对象:

问题: 用直接量创建，会产生大量重复的代码，不便于维护

解决: 用构造函数:

构造函数: 描述一类对象统一结构的函数

何时: 只要反复创建多个相同结构的对象时，都要先用构造函数定义一类对象的统一结构。

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做4件事:

1. 创建新的空对象

2. 让新的空对象继承构造函数的原型对象

3. 调用构造函数:

1. 设置构造函数中的this->新的空对象

2. 通过强行赋值的方式，为新对象添加新属性和方法

4. 返回新对象的地址保存到变量中

总结: 构造函数2个角色:

1. 图纸: 描述一类对象的统一结构

2. 装修队: 在空对象中添加固定的属性和方法

js中对象的本质:

js中的对象其实就是关联数组。只不过比关联数组使用简单而已。

vs 关联数组: 1. 下标: obj["成员名"] <=> obj.成员名

2. 都可随时添加新属性

3. 遍历: for in

优: 代码重用

问题: 每个对象实例都重复创建了一个完全相同的函数对象——浪费内存

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: 代码重用，节约内存

何时: 如果多个子对象中拥有相同的属性值或方法定义，其实只需要将相同的属性和方法定义集中定义在父对象中一份即可。所有子对象公用！

如何: js中的继承都是继承原型对象

原型对象: 集中存储统一类型的子对象共有属性和方法的父对象

如何获得: 买一赠一

构造函数.prototype -> 所有子对象的父对象

如何向构造函数中添加共有属性和方法:

构造函数.prototype.属性名=值;

构造函数.prototype.方法名=function(){...}

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，所有子对象共用的属性

获取属性值: 对象.属性名 相同

修改属性值: 修改自有属性: 对象.属性名=值

修改共有属性:

错误: 对象.属性名=值

正确: 原型对象.属性名字=值

原型链:

什么是: 由多级父对象逐级继承形成的链式结构

保存着: 所有对象的属性和方法

控制着: 对象成员的使用顺序:

优先在对象本地找自有属性使用

自己没有，才延原型链向父对象查找

vs 作用域链: 由多级作用域对象逐级引用形成的链式结构

保存着: 所有的变量

控制着: 变量的使用顺序:

优先在函数作用域对象AO中查找使用

局部没有，才延作用域链向父级作用域查找

凡是不加.的变量，都在作用域链中找

凡是用.才能访问的属性，都在原型链中找

内置对象的原型对象:

每种内置类型其实都有一个构造函数和一个原型对象

创建对象: 自动调用内置对象的构造函数

比如: var arr=new Array();

var now=new Date();

var str=new String();

所有该类型的API其实都保存在构造函数的原型对象中

比如: Array.prototype中: sort() splice() push()...

Date.prototype中: getXXX() setXXX()

toLocaleString()...

解决兼容性问题:

问题: 新的API，就浏览器无法使用！

解决: 2步:

1. 判断当前类型的原型对象中是否包含想用的API

if(typeof 类型.prototype.API !=="function")

2. 向当前类型的原型对象中添加一个自定义的同名API

类型.prototype.API=function(){

... ...

}

多态:

什么是: 同一个事物在不同情况下表现出不同的状态

包括: 重载(overload)和重写

重写(override):

什么是: 如果子对象觉得从父对象继承的成员不好用，可在子对象本地定义同名自有成员，覆盖父对象的成员

为什么: 父对象继承来的成员，不总是想要的

何时: 只要子对象觉得从父对象继承的成员不好用

如何: 只要在子对象中定义和父对象中同名的成员即可

自定义继承:

什么是: 修改一个对象的父对象

何时: 只要父对象不是想要的，就可以修改继承关系，继承另一个对象

如何: 3种:

1. 仅修改一个对象的父对象

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

问题: \_\_proto\_\_是内部属性

解决: Object.setPrototypeOf(child,father)

2. 修改构造函数的原型对象，批量修改所有子对象的父对象

构造函数.prototype=father

时机: 在创建子对象之前

3. 两种类型间的继承:

何时: 如果多个类型之间存在相同的属性结构和方法定义时，就可抽象出一个公共的父类型

为什么: 代码重用和节约内存

如何:

1. 抽象出公共父类型:

父类型的构造函数中定义相同的属性结构

父类型的原型对象中定义相同的方法

2. 让子类型继承父类型:

让子类型的原型对象继承父类型的原型对象

Object.setPrototypeOf(

子类型的原型,父类型的原型

);

在子类型的构造函数中借用父类型的构造函数

错误: 直接调用: 父类型构造函数(参数值...)

原因: 直接调用函数时，函数中的this默认指window，不是当前新对象.

正确: call

父类型构造函数.call(正确的this, 其它参数...);

什么是call: 专门用于替换函数内不正确的this为想要的对象。

何时: 只要函数中的this不是想要的，就可用call换成任意对象。