正课:

\*\*\*\*OOP

封装

\*\*\*\*\*继承

多态

什么是OOP: 程序中都是用对象来描述现实中一个具体事物

什么是对象: 封装一个事物的属性和功能的程序结构

为什么OOP: 便于大程序的维护

何时OOP: 今后写程序都用OOP

\*\*\*面向对象三大特点: 封装，继承，多态

封装:

什么是: 将一个事物的属性和功能集中定义在一个对象中

为什么: 便于大程序的维护

何时: 只要使用OOP，都要先将事物的属性和功能封装在一个对象中，再反复使用对象的功能。

如何: 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

属性和方法统称为对象的成员

3种:

1. 直接量:

var obj={

属性名: 属性值,

... : ... ,

方法名 (){

... this.属性名 ...

},

...

}

何时: 在创建对象时，就知道所有成员

\*\*\*\*\*什么是this: 引用正在调用函数的对象的关键词

简单说: 就是点前的对象

为什么: 方法中不加this的变量，默认仅在作用域链中找，不会去对象中查找。

何时: 只要对象的方法向使用自己的属性时，必须加this

2. 用new创建:

var obj=new Object();//{}

obj.属性名=值;

obj.方法名=function(){

... this.属性 ...

}

何时: 在创建对象时，暂时不知道对象的成员

强调: js中的对象即使创建完成，仍然可以添加新成员

\*\*\*js中一切对象底层都是关联数组！

obj.属性 => obj["属性"]

console.dir(obj) 存储结构和关联数组是完全一样的

简写: new Object() => new Object 或 Object()

问题: 以上两种封装方式仅适合封装一个单独的对象

解决: 构造函数

3. 构造函数:

什么是: 描述一类对象统一结构的函数——图纸

何时: 只要反复创建多个相同结构的对象时，都要先定义统一的构造函数，再使用构造函数反复创建对象

如何: 2步:

1. 定义构造函数描述统一结构:

function 类型名(属性参数){

this.属性名=属性参数;

this.方法名=function(){

... this.属性 ...

}

}

2. 用new调用构造函数，传入具体对象的属性值

var obj=new 类型名(属性值);

其中: new4件事:

1. 创建一个空对象

2. 让新对象继承构造函数的原型对象:

设置新对象的\_ \_proto\_ \_ -> 构造函数的prototype

3. 用空对象调用构造函数: 将构造函数中this->新对象

4. 返回新对象的地址

优: 代码重用

问题: 无法节约内存

解决: 继承:

如何访问对象的属性和方法:

obj.属性 -> 用法和普通变量完全一样

obj.方法() -> 用法和普通函数完全一样

方法也是属性，只不过保存的值是一个函数而已

\*\*\*\*\*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: 今后，只要多个子对象，拥有相同的属性值和功能时

都要将相同的属性和功能集中保存在父对象中一份即可

所有子对象共用！

如何: js中的继承都是继承原型对象:

什么是原型对象: 专门集中存储一类子对象相同属性值和功能的父对象

何时: 今后只要一类子对象共有的相同属性值和功能都要定义在原型对象中

如何: 买一赠一: 每创建一个构造函数，都会自动赠送一个原型对象

用new创建子对象时，会自动设置子对象的\_\_proto\_\_继承构造函数的prototype

如何向原型对象中添加共有成员:

构造函数.prototype.属性=值;

构造函数.prototype.方法=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，被所有子对象共享的属性

获取时: 都可用对象.属性方式

赋值时: 自有属性, 必须: 对象.属性=值

共有属性, 必须: 构造函数.prototype.属性=值

鉴别自有还是共有:

自有: var bool=obj.hasOwnProperty("属性名")

判断"属性名"是否是obj的自有属性

共有: 不是自有，且"属性名" in obj

其中: in: 判断obj自己或obj的父对象中是否包含"属性名"。只要自己或父对象中包含，就返回true。

内置对象的继承关系:

1. 凡是可以new的类型，都是构造函数

2. 每个内置对象的构造函数都对应一个内置的原型对象

3. 内置类型的原型对象中保存着该类型所有子对象共用的API

可解决浏览器兼容性问题:

问题: 如果一个API新的浏览器支持，旧浏览器不支持！

解决: if(typeof 内置类型.prototype.API!="function")

内置类型.prototype.API=function(){

... this //将来调用API的.前的对象

}

原型链: 什么是: 由多级父元素逐级继承形成的链式结构

保存着: 所有的对象成员(属性和方法)

vs 作用域链: 保存着: 所有的变量

控制着: 对象的成员的使用顺序:

优先使用自有的。自己没有，才延原型链向父级查找。

原型链的顶端一定是Object.prototype

vs 作用域链: 控制着变量的使用顺序:

优先使用AO中的局部变量。

局部没有，才去延作用域链向父级作用域查找。

作用域链的终点: window

简单概括:

所有不需要"对象."访问的变量都保存在作用域链中

所有必须用"对象."访问的成员都保存在原型链中

鄙视题: 判断一个对象是不是数组? 有几种办法

0. typeof 只能识别原始类型，function和object

无法进一步识别object中的不同对象类型

1. 验证原型对象:

//如果一个对象的原型对象是Array.prototype

//1. Object.getPrototypeOf(obj)==Array.prototype

2. var bool= Array.prototype.isPrototypeOf(obj)

判断father是否是child的父对象

2. 验证构造函数:

//如果一个对象的构造函数是Array

//1. obj.constructor==Array

2. var bool=obj instanceof Array

判断obj是否由构造函数Array创建出来

instance: 实例: 一个类型中的一个具体的对象

实例化: 用new创建一个对象

问题: 验证不够严格:

即使创建时不是使用数组创建的，但是只要原型链上有数组类型，也认为是数组。

解决:

3. 检查内部属性class:

class是每个对象中记录对象创建时使用的类型的属性

一旦对象被创建，class属性就无法被修改！

如何获得class:

唯一的办法: 调用Object.prototype中的toString()

输出结果: "[object Object]"

引用类型的对象 Class

问题: 几乎所有内置对象的原型对象都重写了Object中的toString方法，所有内置对象的子对象，都无法直接调到Object的toString

解决: call

任意对象.任意方法.call(替换的对象)

多态: 同一个事物，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，可以在子对象本地定义同名成员，覆盖父对象的成员。

为什么: 体现子对象与父对象之间的差异

何时: 只要子对象觉得父对象中的成员不好用，就可在本地重写父对象的成员。

自定义继承:

1. 仅设置两个对象间的继承关系:

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

问题: \_\_proto\_\_也是内部属性, 有可能被浏览器禁用

解决: API: Object.setPrototypeOf(child,father);

设置child继承father

2. 批量设置多个子对象的继承关系

只要修改构造函数的prototype对象即可

构造函数.prototype=father

时机: 定义完构造函数之后，立刻修改原型对象

3. 两种类型间的继承:

何时: 如果发现多个类型之间拥有相同的属性结构和方法时

就要抽象出一个父类型

如何: 1. 定义父类型:

父类型的构造函数，集中定义相同的属性结构

父类型的原型对象，集中定义相同的方法

2. 让子类型原型对象继承父类型原型对象

结果: 保证子对象可调用抽象父类型原型对象中的方法

3. 在子类型构造函数中借用父类型构造函数

结果: 请父类型构造函数帮助构建共有的属性结构

问题: 直接调用父类型构造函数, 其中的this默认指window。

解决: call -> 只要this不是想要的！就可用call随便替换

父类型构造.call(this, 参数。。。)