正课:

1. \*\*\*面向对象:

继承, 多态

2. \*\*\*\*ES5

1. \*\*\*面向对象:

问题: 放在构造函数中的方法定义，会被反复创建副本，浪费内存！

解决: 继承

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！

何时: 只要多个子对象，需要相同的属性值或功能时

如何: js中的继承都是自动继承原型对象

原型对象:

什么是: 存储同一类型的所有子对象共有成员的父对象

何时: 只要多个子对象需要相同的属性值或功能时，都应该集中定义在原型对象中

如何使用: 2步:

1. 创建: 自动创建！买一赠一！

只要创建一个构造函数，都附赠一个空的原型对象

2. 继承: 自动继承！

只要使用构造函数创建一个子对象时，都会让子对象自动继承构造函数的原型对象——new的第2 步

如何向原型对象中添加共有成员:

类型名.prototype.新成员=值/function(){...}

成员的访问顺序:

优先使用自有成员

自己没有，才去父级原型对象中查找

自有属性和共有属性:

自有属性: 保存在对象本地，归当前对象独有的属性

共有属性: 保存在公共的父级原型对象中，所有子对象共享的属性

获取时: 没有差别: 子对象.属性名

修改时: 自有属性: 子对象.属性名=值

共有属性必须通过原型对象修改:

类型名.prototype.共有属性名=值

判断是否自有属性:

var bool=obj.hasOwnProperty("属性名")

内置类型的原型对象:

其实每种类型，都有两部分组成:

构造函数: 专门负责创建子对象

构造函数的原型对象: 专门负责保存所有子对象共有的属性和API

问题: 旧浏览器不支持新的API或浏览器中的API不够用！

解决: 为浏览器添加想要的API

如何: 2步:

1. 先判断当前浏览器是否支持该API:

其实就是判断当前浏览器的指定类型的原型对象中是否包含想用的API

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

2. 向指定类型的原型对象中添加API

类型名.prototype.API=function(参数列表){

... ...

}

原型链:

什么是: 由多级父元素逐级继承，形成的链式结构

作用: 存储必须用.才能访问的所有对象可用的成员

控制成员的使用顺序和使用范围

顺序: 先自有，再共有

使用范围: 原型链中越靠上的成员，共享范围越大

顶级父类型: Object, 原型对象: Object.prototype

放在Object.prototype中的成员，所有对象都可使用

比如: toString() getOwnProperty()

vs 作用域链: 存储所有不用.就可直接访问的变量

控制着变量的使用顺序: 先局部，后全局

顶级作用域是: window——全局作用域对象

鄙视: 判断一个对象是不是数组类型！共几种方式!

错误: typeof: 只能区分原始类型和函数

无法进一步细致区分对象的类型名

正确:

1. 判断爹(原型对象)

var bool=父对象.isPrototypeOf(子对象)

2. 判断妈(构造函数)

obj.constructor===构造函数

问题: constructor是隐藏属性，不推荐使用:

变通: obj instanceof 构造函数

强调: 以上两种方式都不仅检查直接父类型，且检查整个原型链。

3. 验DNA:

其实,每个对象内都隐藏着一个class属性

class属性保存了对象创建时的最初类型

不随继承关系改变而改变

错误1: obj.class 拿不到，返回undefined

唯一正确办法: 只有Object.prototype中最原始toString() 才能输出class

返回值: [object class属性值]

Object

Array

Date

错误2: 使用子对象.toString()，容易被父对象中的toString()重写

解决: 用call强行调用！

Object.prototype.toString.call(obj)

//在运行时临时相当于obj.toString()

4. isArray()

专门判断任何一个对象是不是Array类型

如何: var bool=Array.isArray(obj)

其实内部使用的就是第三种方式——严格的验证！

鄙视: 何时将函数定义在原型对象中，何时将函数直接定义在构造函数上

答: 如果只允许指定类型的子对象才能使用的函数，必须放原型对象中，继承使用

如果希望不限制类型，所有对象都能使用的函数，可以直接放在构造函数上

比如: sort() push() isArray()

Array.prototype.sort()

Array.prototpye.push()

Array.isArray()

多态: 同一个函数在不同情况下，表现出不同状态

2种:

1. 重载overload:

2. 重写override:

什么是: 如果子对象觉得父对象成员不好用！可在子对象本地定义同名成员，覆盖父对象成员。

为什么: 因为从父对象继承来的成员不一定都是好用的！

何时: 只要从父对象继承来的成员不好用，就可以重写自己的!

如何: 只要在子对象本地定义同名成员

原理: 成员的使用顺序: 先自有，再共有

自定义继承关系:

1. 只修改一个对象的父对象:

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

问题: \_\_proto\_\_是内部隐藏属性

解决: Object.setPrototypeOf(child,father)

2. 修改所有子对象的父对象:

其实就是修改构造函数的原型对象:

构造函数.prototype=father;

3. 两种类型间的继承:

问题: 两种类型间拥有部分相同的属性结构和方法定义

解决: 定义抽象父类型

如何: 3步:

1. 定义抽象父类型:

父类型构造函数: 保存所有子类型中共有的属性结构

父类型原型对象: 保存所有子类型中共有的方法定义

2. 在子类型原型对象中借用父类型构造函数

错误: 直接调用!

原因: 如果不用new , 直接调用构造函数, 其中this默认都指window！！！

正确: 只要函数调用时，this不是想要的，都用call替换！

父类型构造函数.call(正确的this,参数......)

3. 让子类型原型对象继承父类型原型对象