正课:

1. \*\*\*OOP

继承:

多态:

2. \*\*\*\*ES5

1. \*\*\*OOP

继承:

自有属性和共有属性:

自有属性: 保存在当前对象本地的，归当前对象独有

共有属性: 保存在父级对象中的属性，归当前类型下所有子对象共用的属性

读取: 两者没有差别!

修改: 共有属性，只能用父级原型对象修改

自有属性，只能用子对象自己修改

内置类型的原型对象:

内置类型: ES标准中规定的，浏览器已经实现的类型

11个:

String Number Boolean —— 包装类型

Array, Date, RegExp, Math

Error

Function Object

Global(全局作用域对象,在浏览器中被window代替)

包装类型:

什么是: 专门保存一个原始类型的值,并提供操作原始类型值的API的类型

为什么: 原始类型的值，本身什么API都没有

比如: var n=345.678; n.toFixed(2)

var c="张"; c.charCodeAt() 24352

何时: 只要试图用原始类型的值调用API时，程序都会自动创建包装类型的对象，来帮助原始类型的值完成任务。

如何: 自动创建，自动调用! 调用后自动释放

n.toFixed(2) => 1. typeof n => number

2. new Number(n).toFixed(2) => 345.68

3. 释放new Number(n)

鄙视: var str="Hello";

str.money=10; //new String(str).money=10

//new String(str)释放！

console.log(str.money);//undefined

//new String(str).money

总结: 其实多数内置类型都由两部分组成:

1. 构造函数: 创建该类型的子对象，保存数据

2. 原型对象: 保存该类型的所有子对象共用的API

特例:

window是一个独立的全局对象，无需创建，也不是构造函数，也没有原型对象

Math也是一个独立的对象。

利用内置类型的原型对象解决API的兼容性问题:

原型链(prototype chain):

什么是: 由多级父元素逐级继承，形成的链式结构

保存: 所有对象中的属性

控制着: 对象属性的使用范围和使用顺序

顺序: 先用自有，再用共有

范围: 处在继承关系越上层的属性，共用范围越广

vs 作用域链：

1. 保存的东西:

作用域链中保存所有不属于任何对象的变量

原型链保存所有对象中的属性

2. 控制着顺序:

作用域链控制着变量的使用顺序: 局部->全局

原型链控制着对象属性的使用顺序和范围: 自有->共用

鄙视: 如何判断一个对象是数组, 共有几种方法?

0. typeof不行! typeof只能区分基本类型，函数和对象。无法进一步区分对象的种类

1. 判断原型对象

Array.prototype.isPrototypeOf(obj)

判断Array.prototype是不是obj的父对象

强调: isPrototypeOf不仅检查直接父级，而是检查整个原型链。

2. 判断构造函数

obj.constructor===Array

问题: constructor是隐藏属性,不建议使用

解决:

obj instanceof Array

实例: 一个类型下的一个具体对象

强调: 不仅检查直接父级，且检查整个原型链

问题: 仅靠继承关系判断对象的类型，不够严谨。

3. 判断内部属性class:

每个对象内部都有一个隐藏的class属性

class属性在创建对象时，就记录该对象的类型名

class属性不会随之后的继承关系改变而改变!

问题: class不能通过.直接访问

解决: 唯一访问对象class的办法是调用顶级Object的原型对象中的toString()

问题: 多数内置类型的原型对象中都重写了toString()，所以底层子对象再没有可能调用顶级toString

解决: .call() 抢: 让一个对象强行调用，本来无法调用到的任何函数。

如何: Object.prototype.toString.call(obj)

强抢Object原型中的toString给obj调用

瞬间: obj.toString()->Object.prototype.toString

总结: 通过class判断对象的类型，不受继承关系的影响

4. 调用Array.isArray(obj)

原理: 同第三种方法

鄙视: 一个函数何时放在原型对象中，何时定义在构造函数对象上

如果一个函数，只限于当前类型的子对象才能调用时，放在原型对象中。

放在原型对象中的方法，必须先创建子对象，再用子对象调用原型对象方法。

如果一个函数，不确定将来调用它的对象的类型，就应该放在构造函数对象上。

放在构造函数对象上的方法，不用创建子对象，直接用构造函数.方法()就可调用。

多态:

什么是: 同一个函数在不同情况下表现出不同的状态

包括: 2种:

1. 重载(overload): 根据传入参数的不同，执行不同的步骤

2. 重写(override):

什么是重写: 子对象觉得从父对象继承来的成员不好用，可在子对象本地定义同名成员，覆盖父对象成员

为什么: 体现父对象和子对象之间的差异！

　　 何时: 只要子对象觉得父对象继承来的成员不好用

如何: 只要在子对象本地定义和父对象成员同名的成员

自定义继承:

1. 仅修改一个对象的父对象:

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

问题: \_\_proto\_\_是内部属性，可能被禁用

解决: Object.setPrototypeOf(child,father)

将father设置为child的父对象

2. 同时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: 两种类型间包含部分相同的属性结构和方法定义

解决: 抽象出一个公共的父类型:

如何: 4步:

1. 定义父类型构造函数，包含相同部分的属性结构:

2. 在父类型的原型对象中添加公共的方法

3. 让子类型原型对象继承父类型原型对象

4. 在子类型构造函数内借用父类型构造函数

错误: 直接调用父类型构造函数

一个函数不用.也不用new调用，其中的this默认都指window。导致fname和speed都泄露到全局

正确: .call，修改函数中的this为正确的对象

何时: 如果调用函数时，函数中的this不是想要的，就可用call强行替换其中的this为指定正确对象

如何: 函数.call(obj,参数,...)

调用函数，并将函数中的this临时替换为obj