正课:

1. OOP

继承

多态

2. ES5

1. OOP

继承:

原型链: 由多级父元素，逐级继承，形成的链式结构

保存了: 对象的属性和方法

控制着: 对象的属性和方法的使用顺序和共享范围

顺序: 先用自有，自己没有，才用共有

共享范围: 越靠原型链上层的共有成员，共享范围越广

vs 作用域链:

保存了所有不加.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

鄙视: 判断一个对象是不是数组类型, 共有几种方法

错误: typeof只能区分原始类型和function

无法进一步区分引用类型对象的具体类型

1. 判断原型对象:

var bool=Array.prototype.isPrototypeOf(obj)

等效于: obj.\_\_proto\_\_==Array.prototype

2. 判断构造函数:

var bool=obj instanceof Array;

等效于: obj.constructor==Array

问题: 不仅检查直接父对象，且检查整个原型链

解决:

3. 检查class属性:

每个对象内，都有一个class属性。

class属性记录了对象创建时的类型名称

class属性不随继承关系改变而改变

问题1: class属性是内部属性, 不允许用.直接访问!

解决: 唯一可查看class属性的办法:

只有Object.prototype.toString() 才可输出:

[object Class]

问题2: 多数内置类型的子对象，无法直接调用Object.prototype中的原始toString()

因为各自类型的原型对象，都纷纷重写了toString()

解决: call 抢:

想调用的函数.call(对象)

在执行时，相当于: 对象.想调用的函数()

Object.prototype.toString.call(obj)==="[object Array]"

4. 直接调用Array.isArray(obj)

强调: 原理同第三种方式，也是严格的检查

鄙视: 一个方法，何时定义在原型对象中，何时定义在构造函数对象上?

实例方法:保存在原型对象中，只能被当前类型的子对象调用的方法。

何时: 如果一个方法，只希望当前类型的子对象才能调用时

如何调用: 必须先创建该类型的子对象，再用子对象.调用实例方法。

比如: arr.sort() arr.indexOf() arr.push()

静态方法: 保存在构造函数对象上，不需要创建子对象，就可用构造函数.直接调用的方法

何时: 如果希望任何类型的对象，都可使用的方法，就可定义为静态方法

如何调用: 不需要创建子对象，只要用构造函数.就可直接调用。

比如: String.fromCharCode(unicode)

Array.isArray(obj)

多态: 同一个方法，在不同情况下表现出不同的状态

包括:

重载overload

重写 override:

什么是: 子对象觉得从父对象继承来的成员不好用时，可在子对象本地定义同名自有成员，覆盖父对象成员。

何时: 只要子对象觉得从父对象继承来的成员不好用！

如何:

自定义继承:

1. 修改一个对象的父对象:

Object.setPrototypeOf(child,father)

等效于: child.\_\_proto\_\_=father

2. 修改多个子对象的父对象:

其实就是修改构造函数的原型对象

时机: 还未开始创建子对象之前

定义完构造函数后，立刻更换

3. 两种类型间的继承:

何时: 只要定义多个类型时，发现拥有部分相同的属性结构和方法时

如何: 2步:

1. 定义抽象父类型:

什么是抽象父类型: 集中存储多个类型间相同的属性结构和方法 的类型

如何定义:

1. 抽象父类型的构造函数中保存相同的属性

2. 抽象父类型的原型对象中保存相同的方法

2. 让子类型继承父类型: 2步:

1. 在子类型构造函数中，先调用父类型构造函数

问题: 直接调用父类型构造函数, 则父类型构造函数中的this默认指window

解决: call 替换函数内不想要的this为指定的对象

2. 设置子类型原型对象继承父类型原型对象

2. ES5:

严格模式:

什么是: 比普通js运行机制，更严格的模式

何时: 今后，所有代码的开发，都必须在严格模式下！

如何: 2种:

1. 在整个<script>或js文件范围内启用严格模式:

在顶部添加: "use strict";

何时: 只要是新项目的开发，都必须整个script或js文件启用严格模式!

2. 仅在一个函数内启用严格模式:

在函数内的顶部加: "use strict";

何时: 旧项目向严格模式迁移时

规定:

1. 禁止给未声明的变量赋值

2. 匿名函数自调和普通函数调用中的this不再默认指向window，而是undefined

都是为了避免内存泄漏

3. 禁用arguments.callee

什么是: 在函数调用时，在函数内部引用当前函数本身

何时: 递归

为什么: 为了在函数内自动获得当前函数名,避免写死

递归的问题: 重复计算量极大！效率极低!

所以: 禁用arguments.callee，等于禁止使用递归！

解决: 所有递归，都可用循环代替!

4. 将静默失败升级为错误!