正课:

1. OOP

2. ES5

1. OOP

静态成员:

什么是: 不需要创建子对象，就可用构造函数直接调用的方法。

何时: 有些函数调用时暂时不确定传入的对象的类型

如何: 2步:

1. 定义时: 将函数直接定义在构造函数对象上

2. 调用时: 无需创建子对象，直接用构造函数.静态方法()调用

总结:

1. 必须某个类型的子对象，才能使用的方法，放在原型对象中——实例方法

2. 不需要具体子对象，就可直接调用的方法，放在构造寒素上——静态方法

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

为什么: 原始类型的值，只是一个值而已，本身不带任何功能

何时: 只要试图对原始类型的值，用.访问属性或调用方法时，自动使用包装类型

如何:

var str="Hello";

str.toUpperCase();

typeof str string

new String(str).toUpperCase()

new String()释放

自定义继承:

1. 仅修改一个对象的父对象:

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

Object.setPrototypeOf(child,father)

//设置child的prototype为father

问题: 只能修改一个子对象的父对象

2. 同时修改多个子对象的父对象:

构造函数.prototype=father

强调: 时机: 定义完构造函数后，创建子对象之前换

问题: 随意！

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: 定义抽象父类型

什么是抽象父类型: 集中存储多个子类型共有属性结构和成员方法的父类型

如何: 2步:

1. 定义父类型: 2步:

1. 定义父类型构造函数，包含相同部分的属性结构

2. 定义父类型原型对象，包含相同部分的方法定义

2. 让子类型继承抽象父类型: 2步:

1. 让子类型的原型对象继承父类型的原型对象

2. 让子类型的构造函数借用父类型构造函数

错误: 直接调用父类型构造函数:

为什么: 直接调用: this->window, 属性都泄露到全局

正确: .call() 为函数打针！

何时: 函数中的this不是想要的！

如何: 函数.call(替换this的对象, 实参1, ... ... )

结果: 用指定的对象，替换函数中不想要的this

2. ES5

ECMAScript JavaScript的核心语法标准

简称ES

版本: 3.1 5 6 ... ...

改进:

1. 严格模式:

什么是: 比普通js运行机制要求更严格的运行模式

为什么: js有广受诟病的缺陷

比如: 1. 可以给未声明的变量赋值，且自动在全局创建

2. 静默失败

3. 普通函数调用的this，默认指向window

4. 递归调用:

问题: 效率极低，重复计算量太大

何时: 今后所有项目都应该在严格模式下运行

如何: 2个范围:

1. 整个js文件/<script>标签内启用严格模式:

在顶部添加: "use strict";

2. 仅在函数内启用严格模式:

在函数内顶部加: "use strict";

要求: 1. 禁止给未声明的变量赋值

2. 将静默失败，升级为错误

3. 普通函数调用中的this不再指向window，而是值为undefined。

4. 禁止使用arguments.callee，其实就是禁止使用递归

解决: 绝大部分递归都可用循环代替