正课:

1. OOP

2. ES5

1. OOP

面向对象三大特点: 封装，继承，多态

封装:

为什么要封装: 构造函数可重用代码和结构定义，但无法节约内存

原理: 放在构造函数内的方法定义，每new一次，都会反复创建副本——浪费内存

解决: 继承

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: 只要发现多个子对象，希望拥有共同的方法定义时

如何: js中都是通过继承原型对象来实现继承关系

原型对象: 集中存储多个子对象，共有成员的父对象

何时: js中只要实现继承，都要使用原型对象

如何:

创建: 不用自己创建。买一赠一!

只要创建一个构造函数，都附赠一个原型对象

构造函数的prototype属性指向原型对象

如何继承: 不用自己继承。

new的第二步: 让新对象自动继承构造函数的原型对象

本质: 设置新对象的\_\_proto\_\_指向父对象

凡是从\_\_proto\_\_指出的引用，都是继承关系

我们做什么: 决定将那些方法集中添加到原型对象中

构造函数中，不再包含任何方法定义

调用对象的方法时:

现在子对象本地找方法，

如果找不到，就延\_\_proto\_\_，向父元素查找

内置对象的原型对象:

内置对象: ES标准中规定的，浏览器厂商已经实现的对象

11个:

String Number Boolean ——包装类型

Array Date RegExp Math

Error

Function Object

Global(浏览器中被window代替)

其实能new的，都是构造函数

每个构造函数都有一个原型对象

原型对象中保存着该类型的所有子对象共用的API

问题: 旧浏览器不支持新的API

解决: 在该类型的原型对象中添加自定义方法

强调: 在方法内，可用this指代将来调用该方法的.前的对象

捷径: 在网上搜索: MDN 类型.prototype.方法名

找Polyfill, 直接复制粘贴即可

包装类型: String Number Boolean

什么是包装类型: 专门保存原始类型的值，并提供操作原始类型值的API

为什么要包装类型: 原始类型的值本身什么功能都没有

何时: 只要用原始类型的值，打.访问方法或属性时

如何: 不用自己用！

比如: var n=345.678; n.toFixed(2);  
 typeof n number

new Number(n).toFixed(2) 345.68

//new Number释放

var str="hello"; str.replace("o","0");

typeof str string

new String(str).replace("o","0");

//new String释放

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

保存着所有对象，以及对象的成员

控制着对象成员的使用顺序:

先自有，再向父级找共用

vs 作用域链:

保存所有孤立的变量, 控制着变量的使用顺序:

先局部，后全局

自有属性 vs 共有属性:

自有属性: 当前对象本地保存的属性

共有属性: 原型链中父级对象中的属性

读取属性值: 子对象.属性名

修改属性值: 自有属性: 用子对象.属性名=值

共有属性: 必须用父对象.属性名=值

鄙视: 判断一个对象是数组, 共有几种方法: 4种

typeof: 只能区分原始类型和function

不能进一步区分引用类型

1. 判断原型对象:

//obj.\_\_proto\_\_==Array.prototype

father.isPrototypeOf(child)

判断father是不是child的父对象

判断数组: Array.prototype.isPrototypeOf(obj)

2. 判断构造函数:

//obj.constructor == Array

child instanceof 构造函数

判断child是否是构造函数new出的子对象

判断数组: obj instanceof Array

问题: 不够严格: 不但检查直接父对象，而且检查整个原型链

解决: 验证对象的class属性

3. 验证对象的class属性:

每个对象都有一个隐藏的class属性(不能用.直接访问)

class属性的值，在创建对象时就确定了，用于保存类型名

class属性不随后天继承关系的改变而改变

如何获得class属性:

唯一的办法: 调用顶级父对象Object.prototype中的toString()

问题: 每种内置类型的原型对象都重写了顶级父对象中的toString()，已经不再返回class属性值

所有子对象无法调用到Object.prototype中的toString()

解决: 抢!

要抢的函数.call(对象)

在执行时: 对象.要抢的函数()

判断对象是不是数组类型:

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

用obj去抢顶级父对象中的toString(),如果返回"[object Array]" 就说明是数组

4. Array.isArray(obj)

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:

1. 重载:

2. 重写(override):

什么是: 子对象定义了和父对象相同的成员

为什么: 从父对象继承来的东西，可能不好用

何时: 只要觉得从父对象继承来的东西，不好用

如何: 在子对象中定义和父对象重名的成员

自定义继承关系: 3种:

1. 仅修改一个对象的父对象:

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

Objext.setPrototypeOf(child,father)

2. 批量修改所有子对象的父对象:

构造函数.prototype=father

时机: 在定义构造函数后，立刻更换

3.