正课:

1. \*\*\*OOP

\*\*\*\*\*继承:

多态:

2. ES5

1. \*\*\*OOP

new: 4件事:

1. 创建空对象

2. 自动设置新对象继承构造函数的原型对象:

child.\_\_proto\_\_=constructor.prototype

3. 调用构造函数

3.1将构造函数中的this指向正在创建的新对象.

3.2 通过强行赋值的方式，为新对象添加新属性和方法

4. 返回新对象的地址给变量

\*\*\*\*\*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

构造函数: 优: 可代码重用

缺: 无法节约内存，反复创建同一个方法的多个副本

何时: 只要同一类型的多个子对象间，拥有相同的属性值或方法定义时，都要用继承的方式来节约内存，重用代码

如何:

原型对象(prototype):

什么是: 专门保存同一类型的所有子对象共有成员的父对象

何时: 只要js中实现继承，都要使用原型对象

今后只要同一类型所有子对象共有的成员，都必须集中保存在原型对象中。

如何:

创建: 不用创建:

买一赠一: 定义构造函数同时，已经附赠了一个原型对象

设置继承关系: 自动设置:

new的第二步: 自动设置新对象继承构造函数的原型对象——原型对象中的成员，子对象无需重复创建就可直接使用

使用原型对象:

向原型对象中添加共有属性或方法

构造函数.prototype.共有成员名=值/function(){ ... }

自有属性和共有属性:

自有属性: 仅当前对象自己所有

共有属性: 保存在原型对象中，被所有子对象共有的属性

获取: 子对象.属性名

修改: 自有属性: 只能用子对象.属性名=值

共有属性: 必须通过原型对象修改:

构造函数.prototype.共有属性=值;

判断是否自有属性:

var bool=obj.hasOwnProperty("属性名")

判断obj中是否包含指定的自有属性

如果返回true，说明指定属性是obj独有的自有属性

如果返回false，2种:

1. 自己没有，但原型中有

2. 自己和原型中都没有

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

包括: 11个

String, Number, Boolean —— 包装(box)类型

Array, Date, Math, RegExp

Error

Function Object

Global(被window代替)

包装类型:

什么是: 专门封装一个原始类型的值，并提供操作原始类型值的API 的对象

为什么: 原始类型的值本身，什么功能都没有。

何时: 只要试图对原始类型的值调用函数时，都会自动使用包装类型

如何: 不用自己使用

自动应用:

每当试图使用原始类型，访问属性或方法时，引擎会自动创建包装类型的对象，保存原始类型的值

调用的方法和属性，其实是包装类型对象提供的

内置对象由2部分组成:

1. 构造函数: 专门用于创建当前类型的子对象

比如: Array, Date, RegExp, String, ...

2. 原型对象: 专门保存当前类型所有子对象共用的API

问题: 旧浏览器不支持新标准的API

原型链:

什么是: 由多级父元素逐级继承形成的链式结构

保存: 所有对象中的属性和方法(都用.访问)

控制着: 属性和方法的使用顺序，共享范围:

共享范围: 原型链中，处于越上层的位置，共享范围越广。

vs 作用域链:

保存着所有变量(不需要.，直接访问)

控制着变量的使用顺序: 先局部后全局

鄙视: 如何判断一个对象是不是数组类型，有几种方式

typeof肯定不行: typeof只能区分原始类型的值,对象,函数。无法进一步区分对象的具体类型

1. 判断原型对象:

如果一个对象的父对象是Array.prototype

var bool=father.isPrototypeOf(obj)

//判断father是否是obj的父对象

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

强调: 不但检查直接父元素，且检查整个原型链。只要任意一级为指定父元素，就是同类型。

2. 判断构造函数:

如果一个对象的原型对象的constructor属性为Array

var bool=obj instanceof Array

//判断obj是否是Array类型的实例

//obj.constructor==Array

强调: 检查整个原型链上的constructor属性。只要有一级constructor是指定构造函数，就是同类型

3. 判断对象的class属性:

每个对象内部都有一个隐藏的class属性

用于记录对象创建时的类型名

在创建对象时，一旦确定，后续不随继承关系改变而改变。

问题1: 无法直接通过.访问class属性

解决: 唯一的获取class的方法:

Object.prototype.toString 输出 "[object class]"

Array

Date

...

问题2: 每种类型的子对象，调用toString()的输出结果各不相同

原因: 多态(重写)，每种类型的原型对象中，都根据自身的需要重写了Object.prototype.toString中原始的toString方法。

解决: 抢

要抢的函数.call(主语)

=> 主语.函数()

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

最严格的判断，不随继承关系改变而改变

4. ES5:

var bool=Array.isArray(obj);

专门判断obj是不是数组类型

强调: 原理就是第三种方式，严格判断

鄙视: 何时将函数定义在原型对象中，何时将函数定义在构造函数上

实例方法: 必须先创建子对象，才能用子对象调用的方法

比如: var arr=[ ... ];

arr.sort() arr.push() arr.reverse()

实例方法，都保存在当前类型的原型对象中

何时: 如果要求，只能当前类型的子对象，才能调用的方法。

静态方法: 不需要创建子对象，就能直接调用的方法

比如: String.fromCharCode()

Array.isArray()

静态方法，都直接定义在构造函数对象上。

何时: 如果不要求使用方法的对象类型时，什么对象都能用!

多态:

什么是: 同一个函数，不同情况下调用，表现出不同的状态

包括: 重载和重写

重写(override):

什么是: 在子对象中定义和父对象同名的成员

为什么: 从父对象继承来的成员，不一定都是好用的！

何时: 只要觉得从父对象继承来的成员，不好用! 就可在子对象中定义同名自有成员。——结果: 每次都优先使用自有的成员，不再使用父对象的成员

如何: 在子对象中定义和父对象同名的成员。

自定义继承: 3种:

1. 只修改一个对象的父对象:

Object.setPrototypeOf(child,father)

设置child继承father

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

2. 同时修改多个子对象的父对象:

构造函数.prototype=father;

强调: 时机: 在创建子对象之前，就更换！

3.