正课:

1. Function:

作用域和作用域链

\*\*\*\*\*闭包

2. \*\*\*OOP

1.Function  
 作用域(scope):

什么是: 变量的可用范围

为什么: 保证内部的变量不会影响/干扰外部的程序

包括: 2种:

1. 全局作用域: window

保存全局变量: 特点: 随处可用，可重复使用!

2. 函数作用域:

保存局部变量: 特点: 仅函数内可用，不可重用!

函数的生命周期:

1. 程序开始执行前:

1. 创建执行环境栈(ECS): 用于记录有多少函数正在被调用

2. 在ECS中记录首先调用的浏览器主程序(main)

3. 创建全局作用域对象window，保存main函数执行所需的所有全局变量

2. 定义函数时:

1. 用函数名定义变量

2. 创建函数对象保存函数的定义，函数名变量通过地址引用函数对象

3. 函数对象的scope属性，指回函数来自的作用域

3. 调用函数时:

1. 在ECS中加入本次函数调用的记录

2. 为本次函数调用，创建函数作用域对象(活动对象AO)

3. 在活动对象AO中保存所有局部变量

4. 让AO的parent属性，指向函数的scope引用的作用域对象

调用过程中，变量的使用顺序:

先在AO中，使用局部变量

局部没有，才去全局找

4. 函数调用后:

1. ECS中本次函数调用的记录，出栈

2. 导致函数作用域对象AO释放

3. 导致AO中局部变量释放

作用域链(scope chain):

什么是: 由多级作用域对象逐级引用形成的链式结构

保存着所有变量

控制着变量的使用顺序: 先局部，后全局

\*\*\*\*\*闭包

什么是: 即重用变量，又保护变量不被污染的一种机制

为什么: 全局变量和局部变量都有不可兼顾的优缺点

全局: 优: 可反复使用, 缺: 随处可用，易被污染

局部: 优: 仅函数内可用，不会被污染

缺: 不可重用!

何时: 今后只要希望一个变量，即可重用，又不会被污染

如何: 3步:

1. 用外层函数包裹受保护的变量和内层函数对象

2. 外层函数将内层函数对象返回到外部

3. 使用者调用外层函数，获得返回的内层函数

鄙视: 画简图:

1. 找受保护的变量

2. 找操作变量的内层函数

缺: 闭包比普通函数占用更多的内存空间

解决: 如果闭包不再使用，应尽快释放!

将引用闭包结构的变量赋值为null

2. \*\*\*OOP:

什么是对象: 程序中描述现实中一个具体事物的属性和功能的程序结构.

什么是面向对象: 程序中都是先用对象来描述现实中的具体事物，再按需调用对象的方法，访问对象的属性值。

为什么: 当要管理的数据非常多时，如果单独管理每项数据，极其不便于维护和使用。且不符合现实。

如何: 面向对象三大特点: 封装，继承，多态

封装:

什么是: 创建一个对象集中保存现实中一个事物的属性和功能

何时: 今后，只要使用面向对象的思想编程，都要先创建一个对象，来存储现实中一个事物的属性和功能

为什么: 便于维护和使用，且更符合现实的意义

如何: 3种:

1. 直接量: var obj={

属性:值,

... : ...,

方法:function(){

... this.属性名 ...

}

}

现实中: 具体事物的属性，会成为对象的属性

具体事物的功能，会成为对象的方法

问题: 对象自己的方法不能直接访问自己的属性

因为: 所有不加.直接访问的变量，只能在作用域链中查找，无法自动进入对象中查找属性。

错误解决: 在属性前加对象名.

问题: 对象名只是普通的变量名，可能发生改变。

正确解决: 在属性前加this.

this: 函数执行时，自动创建的，引用正在调用该函数的.前的对象本身。

何时: 只要对象自己的方法，想使用对象自己的成员时，都必须加this.

如何访问对象的成员:

成员=属性+方法

访问属性: obj.属性名

调用方法: obj.方法名()

2. 用new: 2步:

1. var obj=new Object(); //{}

2. obj.属性名=值

obj.方法名=function(){ ... this.属性名 ... }

本质: 起始js对象底层，就是一个关联数组

问题: 每次只能创建一个对象，如果反复创建多个对象，重复的结构代码会非常多！——DRY

解决:

3. 用构造函数反复创建多个相同结构的对象:

构造函数(constructor): 描述一类对象，统一结构的函数

何时: 只要反复创建多个相同结构的对象前，都要先定义构造函数。

如何: 2步:

1. 定义构造函数，描述一类对象的统一结构:

function 类型名(属性参数列表){

this.属性名=属性参数;

this.方法名=function(){

... this.属性名 ...

}

}

2. 调用构造函数反复创建同一类型的对象:

var obj=new 类型名(属性值列表);

new做了4件事:

1. 创建一个新的空对象:

2. 设置新对象的\_\_proto\_\_属性，继承构造函数的原型对象

3. 调用构造函数，并将其中this自动指向新对象

构造函数用强行赋值的方式，将构造函数中规定的属性和方法，添加到新对象中

4. 返回新对象地址

总结: 构造函数两个作用:

1. 描述一类对象的统一结构

2. 将一个新的空对象，装修成规定的格式和内容

优: 代码重用!

问题: 在构造函数中添加的方法，每调用一次构造函数，都会创建该方法的副本。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: 只要多个子对象都拥有相同的成员时，应该仅在父对象中保存一份即可。所有子对象共用!

如何: js中都是继承原型对象

原型对象(prototype): 专门保存同一类型的所有子对象，共有成员的父对象。

何时: 只要多个子对象都拥有相同的成员时，应该仅在原型对象中保存一份即可。

如何:

创建: 不用自己创建，而是买一赠一:

每定义一个构造函数，都附赠一个空的原型对象

继承: 不用自己设置

每new一个子对象，自动设置子对象继承构造函数的原型对象！

共有属性和自有属性:

共有属性: 保存在原型对象中，所有子对象共有的属性

自有属性: 保存在子对象中，仅归当前子对象独有的属性

获取: obj.属性名

修改: 自有属性: 子对象.属性名=值;

共有属性: 只能用 原型对象.共有属性=值

内置对象的原型对象:

内置对象: ES标准中规定的，浏览器已经定义好的对象。

ES标准内置对象: 11个:

String Number Boolean —— 包装类型

Array Date Math RegExp

Function Object

Error

Global(在浏览器中被window代替)

鄙视: 包装类型(box)：

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

为什么: 原始类型的值，仅是一个值而已，没有任何的方法。

何时: 只要试图对原始类型的值调用方法时，都会自动创建对应包装类型的对象。

一种类型: 都包含2部分:

1. 构造函数: 创建当前类型的子对象

2. 原型对象: 保存当前类型所有子对象共有成员

可解决浏览器兼容性问题:

问题: 旧浏览器，无法使用新的API

解决: