正课:

1. \*\*\*Function

2. \*\*\*OOP

1. \*\*\*Function

作用域（scope）：

什么是: 一个变量的使用范围

为什么: 避免函数内外的变量间互相影响

包括: 2种:

1. 全局作用域: window

保存着全局变量: 随处可用，可反复使用

缺点: 不会被释放，造成全局污染和内存泄漏

2. 函数作用域: actived object

保存着局部变量: 仅函数内可用，不可重复使用

缺点: 不可重复使用

原理:

1. 程序开始执行前

创建执行环境栈ECS(Execute Context Stack)

ECS: 用于记录当前正在调用的函数

要调用函数时，函数进入ECS

函数调用后，会从ECS中弹出

调用浏览器的主程序main()——将main()加入ECS

main()会创建window对象，保存内置的对象和方法

只要浏览器不关闭，main()和window会始终存在

2. 定义函数时:

用函数名创建全局变量,保存在window中

创建函数对象，保存函数的定义

让函数名变量引用函数对象

函数对象的scope属性指回函数来自的作用域对象

3. 调用函数时:

函数调用的语句进入ECS

为本次函数调用创建函数作用域对象

在函数作用域对象中创建局部变量

让函数作用域对象的parent属性指向函数来自的作用域

变量的使用顺序: 就近

先局部，后全局

4. 函数调用后

ECS中本次函数调用的语句出栈

导致本次函数调用的函数作用域对象释放

导致局部变量一同释放！

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

作用: 保存所有变量， 控制着变量的使用顺序

\*\*\*\*\*闭包:

什么是: 即重用变量，又保护变量不被篡改的一种程序结构

为什么: 全局变量和局部变量都有缺点:

全局变量: 优: 可反复使用，随处可用

缺: 易被篡改，或造成内存泄露

局部变量: 优: 仅函数内可用，不会被篡改

缺: 不可重用

何时: 想即重用变量，又保护变量

如何: 3步:

1. 用外层函数包裹受保护的变量和内层函数

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，获得内层函数对象

闭包如何形成: 外层函数的作用域对象无法释放

缺点: 1. 比普通函数多占用存储空间

2. 永远无法自动释放，导致内存泄露

解决: 一旦闭包不再使用，要将引用内层函数的全局变量赋值为null -> 导致内层函数释放-> 导致外层函数的AO释放

鄙视: 画简图:

1. 找受保护的变量:

2. 找外层函数抛出了几个内层函数对象

一次外层函数调用抛出的多个内层函数对象，共用同一个受保护的变量

2. \*\*\*OOP

什么是面向对象: 程序中，都是用对象结构，来描述现实中一个具体事物

什么是对象:

用途: 专门描述一个事物的属性和功能的程序结构

本质: 程序中保存一个事物的属性和功能的存储空间

为什么: 在保存大量数据时，便于维护

何时: 所有程序都使用面向对象形式

如何: 三大特点: 封装，继承，多态

1. 封装: 创建一个对象，保存一个事物的属性和功能

事物的属性会成为对象中的属性

事物的功能会成为对象中的方法

如何: 3种:

1. {}: var obj={

属性: 属性值,

... : ... ,

方法: function(){

... ...

}

}

访问对象中的成员(属性+方法)

访问属性: 对象.属性 用法和普通变量的用法一样

使用方法: 对象.方法() 用法和普通函数的用法一样

问题: 对象自己的方法，不能直接使用自己的属性名:

为什么: 默认情况，引擎不会自动进入对象中找属性，只能在作用域链中找普通的变量

不好的解决: 在方法中，属性名前写死对象名

问题: 如果对象名变化，则内部必须跟着变化，否则出错！

正确的解决: this  
 this: 函数中自动创建的，指向正在调用函数的.前的对象

总结: 今后对象自己的方法，要使用自己的属性，必须加this.

2. 用new: 2步:

var obj=new Object();

obj.属性名=值;

强调: js中的对象，可随时通过强行赋值的方式，添加新属性

揭示: 其实js中一切对象底层都是关联数组

访问对象成员: 对象.属性 对象["属性"]

特殊: 如果属性名来自于变量或动态生成

就只能用对象.[变量]

问题: 每次只能创建一个对象

解决: 用构造函数反复创建多个相同结构的对象

3. 用构造函数创建:

什么是: 描述一类对象统一结构的函数

为什么: 重用对象结构的定义

何时: 只要反复创建多个相同结构的对象时

如何: 2步:

1. 定义构造函数:

function 类型名(参数, 参数, ...){

this.属性名=参数;

this.属性名=参数;

... = ... ;

this.方法名=function(){

... this.属性名 ....

}

}

2. 调用构造函数创建新对象:

var obj=new 类型名(参数值, 参数值, ... )

new: 4件事:

1. 创建一个空对象

2. ?

3. 用新的空对象，调用构造函数

将this自动指向新对象

通过强行赋值的方式，给新的空对象添加新属性

4. 将新对象地址返还给obj