正课:

1. Function

\*\*\*\*作用域和作用域链

\*\*\*\*\*闭包

2. \*\*\*OOP

封装

继承

多态

1. Function

函数声明周期: 4个阶段

1. 程序开始执行前:

创建执行环境栈(数组,ECS): 依次保存正在调用的函数记录的数组

浏览器启动时，自动调用浏览器主程序main()，放入ECS中

main()因为自身的需要，创建并引用全局作用域对象window

将来所有在全局范围声明的变量和函数，默认都保存在window中，可反复使用！

2. 定义函数时:

用函数名创建变量

创建函数对象保存函数定义

函数名变量引用函数对象

函数对象用隐藏的scope属性引用window

3. 调用函数时:

只要调用一个函数，必须向ECS中添加本次函数调用的记录

为本次函数调用创建函数作用域对象——活动对象(AO)

在活动对象中创建所有局部变量

活动对象自动引用函数的scope指向的父级作用域

变量的使用顺序:

优先使用活动对象中的局部变量

局部没有，才去全局找

4. 函数调用后:

将本次函数调用的记录从ECS中删除

导致: 活动对象无人引用，被释放！

导致: 局部变量异同释放！

——局部变量在函数调用后，不可重用！

作用域链(scope chain): 由多级作用域对象逐级引用，形成的链式结构

\*\*\*\*\*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: 优: 可重用, 缺: 易被污染

局部变量: 优: 不会被污染, 缺: 不可重用

何时: 即重用变量，又保护变量不被污染

如何: 3步:

1. 用外层函数包裹受保护的变量和内层函数

2. 外层函数将内层函数返回到外部

3. 调用外层函数获得内层函数对象

鄙视: 闭包如何形成:

外层函数调用后，外层函数的函数作用域对象无法释放

闭包缺点: 闭包中的函数比普通函数占用更多内存空间

解决: 一旦不再使用闭包，应及时释放闭包:

将保存内层函数的变量赋值为null

何时: 只要为一个函数，添加持久的专属的变量时

鄙视: 画简图:

1. 找受保护的变量(外层函数的局部变量), 确定其在外层函数调用后的最后值被改为多少。

2. 找外层函数共向外抛出了几个内层函数:3种:

1. return

2. 直接给全局变量赋值一个内层函数

3. 将内层函数包裹在数组或对象内，返回数组或对象。

3. 同一次外层函数调用，返回的多个内层函数，共用同一个受保护的变量

多次外层函数调用，返回的多个内层函数，各自使用自己的受保护的变量。不会互相干扰。

2. \*\*\*OOP

什么是对象: 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: 程序都是用对象结构来描述现实中一个具体事物的属性和功能

为什么: 便于大量数据的分类管理和维护

何时: 今后，只要管理多个事物的程序，都要用面向对象编程

如何: 三大特点: 封装，继承，多态

1. 封装:

什么是: 将一个事物的属性和功能集中定义在一个对象中

为什么: 便于多个事物的数据的使用和维护，避免相互干扰

何时: 面向对象的第一步，都是用对象来封装事物的属性和功能，再按需使用对象的属性和功能。

如何创建对象: 3种:

1. 直接量: {}

var obj={

属性: 值,

... : ... ,

方法: function(){

...

}

}

问题: 对象的方法，如何访问对象自己的属性

错误解决: 在方法中直接使用属性名作为变量

因为: 不加.的变量，默认只能在作用域链中查找

无法自动进入对象中

解决1: 在方法中的属性名前加"对象."前缀——必须

问题: 对象名可能发生变化

正确解决: 在方法内，用this关键词代替写死的对象名

什么是this: 函数内 自动引用正在调用函数的.前的对象 的关键词。

何时: 只要对象自己的方法，要访问对象自己的属性，必须用"this."

如何访问对象的成员: (成员=属性+方法)

访问属性: 对象.属性名 对象属性的用法和普通变量完全一样！

调用方法: 对象.方法名() 对象方法的用法和普通函数完全一样!

2. 用new:2步:

var obj=new Object();//创建空对象

obj["属性"]=值; //如果属性名是动态变化的，则必须用[变量]方式访问属性。不能用.

obj["方法"]=function(){

... this.属性名 ...

}

揭示: js中一切对象底层其实都是关联数组

进一步揭示:

1. js中的对象，即使创建后，也可随时添加新属性和功能

2. js中访问对象中不存在的属性，不报错，而是返回undefined

3. 用for...in...遍历对象每个属性

问题: 一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: 2步:

1. 定义构造函数:

什么是: 描述同一类型的所有对象的同一结构的函数

为什么: 重用结构, 减少代码冗余

何时: 只要需要反复创建多个相同结构的对象时，都要先定义构造函数

如何:

function 类型名(属性参数列表){

this.属性名=属性参数;

this. xxx = xxx ;

this.方法名=function(){

... this.属性 ...

}

}

2. 调用构造函数反复创建多个相同结构的对象:

var obj=new 构造函数名(属性值列表);

其中new: 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,并将构造函数中的this指向正在创建的新对象

this.属性=属性参数;

本意: 将属性参数值保存到this->new新对象中的指定"属性"

但是: 因为新对象什么属性都没有

所以: 每条this.属性=属性参数; 都是在新对象中强行创建新属性。—— 所以才成为"构造函数"

4. 将新对象地址返回给变量中保存

问题: 每个对象的方法都会被重复创建——浪费内存！

解决: 继承

2. 继承:

什么是: 父对象的成员，子对象无需创建就可直接使用

为什么: 代码重用, 节约内存！

何时: 只要多个子对象，拥有相同的属性值或方法定义时，都要只在父对象中保存一份即可。所有子对象共用！

如何: js中的继承都是继承原型对象

什么是原型对象: 集中存储同一类型的所有子对象，共有成员的父对象.

何时: 只要保存多个子对象共有的成员，都要放在原型对象中集中定义一份即可

如何:

创建: 不用手动创建

在定义构造函数时，附赠一个原型对象

设置继承关系: 不用手动设置!

在new的第二步，让新对象自动继承构造函数的原型对象

将共有的成员，放入原型对象中集中存储1份：

构造函数.prototype.成员名=值/function(){...}

3. 多态