正课:

1. 错误处理:

2. \*\*\*\*Function

1. 错误处理:

错误: 程序执行过程中，导致程序无法正常执行的现场

bug:

现象: 程序一旦出错，默认会报错，并强行退出

错误处理: 即使程序出错，也保证程序不会强行退出的机制

如何:

try{

可能出错的语句

}catch(err){//只在出错时，才执行

错误处理代码: 提示用户, 记录日志, 保存数据

//其中: err: 错误对象

//什么是: 错误发生时，自动创建的，记录错误信息的对象

}

效率:

问题:

1. 放入try catch中的代码，即使不出错，效率也比正常代码略低。

2. 程序出错都会创建错误对象，占用更多内存

解决: 如果可提前预知错误原因，则建议用if...else代替try catch

优: 1. 不影响程序执行效率，2. 节约内存

主动抛出错误:

什么是: 在本来程序不出错的情况下，为了以防万一，主动创建一个错误，并抛出。

何时: 在协作开发中，函数的定义者向函数的使用者抛出错误，用以告知对方错误的使用了函数。

如何: throw new Error("错误信息")

js中错误的类型:

6种:

SyntaxError 语法错误

ReferenceError 引用错误: 要找的变量不存在！

TypeError 类型错误: 错误的调用了API或错误的使用了()或[]

RangeError 范围错误: 参数的超范围

URIError EvalError

2. \*\*\*\*Function

创建函数: 3种:

1. 声明:

function 函数名(参数列表){函数体; return 返回值}

问题: 会被声明提前

解决:

2. 直接量:

var函数名=function (参数列表){函数体; return 返回值}

优: 不会被声明提前

揭示: 函数名其实仅是一个变量

函数定义其实是一个引用类型的对象

函数名通过函数对象的地址引用着函数

3. 用new:

var函数名=

new Function("参数",...,"函数体; return 返回值")

强调: 无论是参数名还是函数体都必须放在""中

比如: var cmp=new Function("a","b","return a-b");

\*\*\*重(chong)载:overload

什么是: 相同函数名，不同参数列表的多个函数，在调用时，程序可根据传入参数的不同，自动调用对应函数执行

为什么: 减少API的个数，减轻调用者的负担

何时: 一项任务，如果根据不同的参数，执行不同的操作流程时

如何:

问题: js默认不支持重载语法: js不允许多个同名函数同时存在

后果: 最后一个函数会覆盖之前所有函数

解决: arguments

什么是: 函数内自动创建的

接收所有传入函数的参数值的

类数组对象: 长的像数组的对象:

vs 数组: 相同: 1. 下标, 2. length. 3. for遍历

不同: 类型不同，API不通用

匿名函数:

什么是: 定义函数时，没有指定函数名的函数

何时: 只用一次的函数，都不要写名

为什么:

1. 节约内存，及时释放

垃圾回收: 垃圾回收器程序会自动释放不被任何变量引用的对象

垃圾回收器: 专门检查并回收不再被使用的对象的小程序

2. 避免产生全局变量污染

如何:

1. 回调: 自己创建函数后，不调用，传递给其它函数，让其它函数使用。

比如: arr.sort(比较器函数)

str.replace(reg,function(kw){...})

绑定事件处理函数

2. 自调: 定义函数后，立刻调用自己

如何: (function(){

...

})();

+function(){

...

}();

何时: 不希望产生任何全局变量污染时，都可将代码封装在匿名函数中，自调执行。

为什么: 划分临时作用域，避免创建全局变量

作用域和作用域链:

作用域scope: 变量的使用范围

js中包括两种作用域:

全局作用域: window

保存: 全局变量

优: 随处可用, 可反复使用

缺: 易造成全局污染

函数作用域:

保存: 局部变量 优: 仅函数内可用，不会被污染

缺: 不可重用

函数的生命周期:

程序执行前: 调用浏览器主程序,创建全局作用域对象window

定义函数:

在全局创建函数名变量

在window外创建函数对象保存函数定义

函数名变量使用地址引用函数对象

函数对象的隐藏属性scope指回函数诞生的作用域window

调用函数:

在ECS中添加本次函数调用的记录

创建本地函数调用所需的函数作用域对象AO(活动对象)

在AO中创建局部变量

AO的隐藏属性parent指向父级作用域对象

变量的使用顺序:

优先使用函数作用域AO中的局部变量

局部没有，才去父级作用域找

调用后:

ECS中本函数调用的记录出栈

导致函数作用域AO没人引用，被释放

导致AO中的局部变量一起释放

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

2个职责:

1. 保存所有变量:

　　 全局变量都保存在window中

局部变量都保存在AO中

2. 控制变量的使用顺序: 先局部，再全局

\*\*\*\*\*闭包:

什么是: 即重用一个变量，又保护变量不被污染的机制

为什么:

全局变量: 优: 可重复使用，随处可用

缺: 易被污染

局部变量: 优: 仅函数内可用，不会被污染

缺: 不可重用！

何时: 只要希望即重用一个变量，又保护变量不被污染，就只能用闭包

如何: 3步:

1. 用外层函数包裹住受保护的变量和内层函数

2. 外层函数将内层函数返回

3. 使用者调用外层函数获得返回的内层函数对象

闭包形成的原因: 外层函数的函数作用域AO无法释放

缺: 比普通函数占用更多的内存空间

建议: 如果闭包不再使用，就要及时释放

如何释放? 将引用内层函数的变量置为null

鄙视: 2步:

1. 先找受保护的局部变量，并确定最终保存的值

2. 再找操作变量的内层函数: 3种情况:

1. 直接return function

2. 给全局变量赋值: 全局变量=function

3. 将内层函数保存在数组/对象内，整体返回

3. 画简图: