正课:

1. \*\*\*\*Function

1. 创建

2. 重载

3. 匿名函数

4. \*\*\*作用域和作用域链

5. \*\*\*\*\*闭包

2. 错误处理

1. \*\*\*\*Function

1. 创建: 3种:

1. 声明:

function 函数名(参数列表){函数体; return 返回值}

特点: 会被声明提前

2. 直接量:

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

特点: 不会被声明提前

何时: 只要希望函数的定义不被声明提前时

3. 用new:

var 函数名=

new Function("参数1","参数2","函数体;return 返回值")

强调: 所有参数变量和函数体，必须用""包裹

鄙视: 以下函数创建正确的是:

function fun(a,b){return a-b}

var fun=function(a,b){return a-b}

var fun=new Function(a,b,"return a-b") XXX

var fun=new Function("a","b","return a-b")

2. 重(chong)载(overload):

什么是: 相同函数名，不同参数列表的多个函数，在调用时可根据参数的不同，自动选择对应的函数执行。

何时: 如果一件事，根据不同的参数，执行不同的逻辑时

为什么: 减少函数的数量，减轻调用者的负担

问题: js语法，默认不支持多个同名函数同时存在！

如何变通:

1. 仅定义一个函数，不要写参数列表

2. arguments: 函数内自定创建的

用于接收传入函数的所有参数值的

类数组对象

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

3. 函数内，根据arguments中参数的个数或类型，动态决定执行不同操作。

问题: 参数是否还需要

答: 需要:

1. 参数用于提示调用者如何正确使用函数

2. 参数名一般都比arguments简洁且见名知义

3. 匿名函数:

什么是: 创建函数时，不指定函数名的函数

何时: 2种:

1. 当一个函数仅使用一次时

2. 划分临时作用域，避免全局污染时

为什么: 1. 节约内存，2. 避免全局污染

如何: 2种:

1. 回调:

比如:

arr.sort(function (a,b){return a-b})

str.replace(reg,function(kw){return 替换值})

btn.addEventListener("click",function(){...})

2. 自调: 定义匿名函数后，立刻调用自己，调用后立刻释放

如何: 2种:

+function(...){...}(...)

(function(...){...})(...)

何时: 只要不希望造成全局污染时，就用匿名函数包裹所有自定义的代码。

4. \*\*\*作用域(scope)和作用域链(scopechain)

什么是作用域: 一个变量可用范围，也是一个变量的实际存储位置。

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: AO

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:

程序开始执行前:

创建执行环境栈(数组),用于保存将要调用的素有函数

在栈中添加第一个函数调用: 浏览器的主程序main()

主程序创建全局对象window

定义函数:

用函数名创建全局变量

创建函数对象封装函数体

函数名变量引用函数对象

函数对象的scope属性(籍贯)指回window

调用函数:

创建函数作用域对象AO(活动对象)

在AO中添加函数中定义的所有局部变量(var 声明的和参数变量)

变量使用顺序: 优先使用AO中的局部变量

AO中有，就不去全局找

AO中没有，才去全局找

调用后:

ECS中的函数调用出栈

导致AO被释放

导致AO中的局部变量一同释放

作用域链: 由各级作用域对象连续引用形成的链式结构

保存着所有变量，且控制着变量的使用顺序

5. \*\*\*\*\*闭包(closure):

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优缺点:

全局: 优: 可反复使用

缺: 随处可用——易被污染

局部: 优: 仅在函数内可用，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，包裹住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. 使用者调用外层函数，获得返回的内层函数对象

鄙视: 闭包如何形成: 外层函数的作用域对象(AO)无法释放

闭包的问题: 比普通函数占用更多内存空间——多的是外层函数的AO

解决: 将引用闭包函数的变量赋值为null

导致内层函数对象先释放

导致外层函数的AO一并释放

鄙视: 画简图:2步:

1. 找受保护的变量:

外层函数的局部变量(var的或参数变量)

确定在外层函数调用后，受保护变量的最终值

2. 找抛出的内层函数:2个途径:

1. return

2. 直接给全局变量赋值

3. 隐藏: 将内层函数放入一个数组/对象中隐藏返回

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。

2. 错误处理:

什么是错误: 程序运行过程中，导致程序无法继续执行的异常情况。

发生错误后，程序会立刻中断退出

什么是错误处理: 即使程序发生错误，也保证不强行退出的一种机制

为什么: 避免程序强行中断，导致极差的用户体验

何时: 只要希望即使程序发生错误，也保证不强行退出

如何:

try{

可能发生错误的语句

}catch(err){//只有发生错误时，才执行

错误处理代码: 1.保存进度; 2. 提示用户; 3. 记录日志

}

其中: err是错误对象:

什么是: 错误发生时，自动创建的，保存错误信息的对象

性能问题: 放入try catch中的代码，可能执行效率会降低，且一旦出错，还要额外创建error对象，占用更多内存空间.

解决: 如果可提前预知错误的原因，可用if...else...来代替try catch——对于开发人员的经验要求非常高

主动抛出错误:

什么是: 当前程序运行出错时，主动新建一个错误，抛出

何时: 在协作开发中，函数的作者，可用主动抛出错误的方式提醒调用者如何正确使用函数。

如何: throw new Error("错误提示");

鄙视: js中错误的类型: 6种

SyntaxError: 语法错误

ReferenceError: 引用错误, 要用的变量未找到

TypeError: 类型错误, 使用错误的类型调用函数或访问元素。

RangeError: 范围错误, 参数超范围

URIError,EvalError;

bug: 只要导致程序出现问题的原因