正课:

1. 错误处理:

2. \*\*\*函数

定义:

重载:

匿名函数:

\*\*\*\*作用域和作用域链

\*\*\*\*\*闭包

1. 错误处理:

错误bug: 程序执行过程中，导致程序无法正常执行的情况

后果: 程序会强行中断退出

错误处理: 即使程序出现错误，也保证程序不异常中断的机制。

如何: try{//始终执行

//可能出错的代码

}catch(err){//只有发生错误时才执行

//err: 错误对象: 发生错误时，自动创建的，封装错误信息的对象

//出错时，执行的错误处理代码:

//比如: 通知用户，记录日志，保存进度

}[finally{//无论是否出错，都必须执行的代码

//一般用于释放资源

}]//可有可无，因为放在try catch之后的代码本来就总是执行的。

问题: 放在try catch中的代码，执行效率会降低

解决: 2种:

1. 尽量少的将代码放入try catch

2. 如果可提前预知错误原因，可用if...else代替try catch，提前预防错误。

抛出自定义异常:

throw new Error("错误信息")

js中错误对象的类型: 6种

SyntaxError: 语法错误: 程序的语法写错了

ReferenceError: 引用错误: 要用的变量没找到

TypeError: 类型错误: 错误的使用对象的类型:

比如: 对非数组使用[]

对非函数使用()

对null/undefined用.xxx

要调用的函数，对象中没有:

比如: console.write();

document.log();

RangeError: 范围错误: 参数值超规定范围:

比如: toFixed(d) d: 0~20之间 如果写-2

URIError, EvelError;

2. Functions:

创建: 3种:

1. 声明: function 函数名(参数列表){

函数体;

return 返回值;

}

问题: 被声明提前

解决:

2. 函数直接量: var函数名=function (参数列表){...}

优点: 不会被声明提前

揭示: 函数是一个引用类型的对象

函数名只是一个引用函数对象的普通变量

3. 用new: 了解

var函数名=new Function("参数","参数","函数体;...")

鄙视: 以下创建函数方式正确的是:

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

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

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

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

重载: overload

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

为什么: 减少API的名字，减轻调用者的负担

何时: 只要一项任务，可能根据不同的参数执行不同的逻辑时。

问题: js语法默认不支持重载

为什么: js中不允许多个相同名称的函数同时存在

后定义的同名函数，会覆盖先定义的。

解决: 2步:

1. 将多个函数合并为一个函数，省略参数列表

2. 函数内，用arguments接住所有传入的变量值

arguments: 每个函数中自动创建的

自动接收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, 类数组对象不能使用数组的API

比如: childNodes children

参数变量依然必要:

1. 参数变量可指导调用者正确的使用函数

2. 参数名都是有意义的，便于维护/提高程序可读性

3. 自定义的参数名通常简洁

匿名函数:

什么是: 定义函数时，不指定函数名的函数

为什么: 节约内存!

何时: 只要一个函数只使用一次时

2种:

1. 回调callback: 将一个函数作为参数传给另一个函数中调用

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

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

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

setInterval/Timeout(function(){...}, ms )

特例: 如果函数可能被反复调用或可能被多个元素共享时，应该用有名的函数。

2. 匿名函数自调:

定义一个函数后，立刻调用该函数，调用后立即释放

何时: 划分临时作用域，避免全局变量污染时

如何: 2种:

1. (function([参数列表]){...})([参数值])

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

\*\*\*\*作用域和作用域链:

作用域(scope): 变量的可用范围

包含2种:

1. 全局作用域对象window

全局变量: 优: 可重复使用，随处可用

缺: 极易造成全局污染

——今后尽量避免使用全局变量

2. 函数作用域对象AO

局部变量: 优: 仅函数内可用，不会造成全局污染

缺: 不可重用！

作用域链(scope chain):

什么是: 由各级作用域对象连续引用，形成的链式结构

函数的声明周期:

1. 程序开始执行前:

程序会创建全局作用域对象window

2. 定义函数时

在window中创建函数名变量引用函数对象

函数对象的隐藏属性scope指回函数来自的全局作用域对象window——就好像人的祖籍

3. 调用函数时

创建本次函数调用时使用的AO对象

在AO对象中添加函数的局部变量

设置AO的隐藏属性parent 指向函数的祖籍作用域对象。——执行时，如果AO中没有的变量可延parnet向祖籍作用域对象找。

4. 函数调用后

函数作用域对象AO释放

导致AO中局部变量释放

作用域链: 2项任务：

1. 保存了所有的变量

2. 控制了变量的使用顺序: 先用局部，局部没有才延作用域链向下查找。

\*\*\*\*\*闭包(closure)

什么是: 即重用变量，又保护变量不被污染的一种结构

为什么: 全局变量: 优: 可重用, 缺: 全局污染

局部变量: 优: 不会被污染, 缺: 不可重用

何时: 只要希望重用一个变量，并保护变量不被污染。

如何: 3步:

1. 使用外层函数包裹受保护的变量和操作变量的内层函数

2. 外层函数将内层函数返回到外部

3. 调用者用外部变量接住返回的内层函数

问题: 闭包结构比普通函数占用更多的内存: 外层函数的函数作用域对象

解决: 使用完闭包后，要主动释放闭包:

将引用内层函数的外部变量置为null

鄙视: 画简图:

1. 找受保护的变量:

是外层函数的局部变量

同时, 在内层函数中被使用

2. 找外层函数都返回了哪些内层函数用于操作受保护的变量: 2种手段:

1. return function

2. 可直接给一个全局变量赋值一个function