正课:

1. Date

2. Error

3. \*\*\*Function

1. Date

什么是: 保存一个时间，提供操作时间的API 的对象

何时: 只要在程序中保存或操作时间都要用Date类型

如何:

创建: 4种:

1. 创建一个日期对象，自动获得当前客户端系统时间:

var now=new Date()

2. 创建一个日期对象，保存自定义时间

var date=new Date("yyyy/MM/dd hh:mm:ss")

3. 用毫秒数创建日期对象:

var date=new Date(ms);

何时: 将数据库中存储的毫秒数，转为程序中可操作的时间对象

如何获得毫秒数: var ms=date.getTime()

4. 复制一个日期对象:

为什么: 日期对象的计算都是直接修改原日期对象。计算后，原日期无法保留

何时: 只要同时保留开始和结束时间时，都要先复制日期对象的副本。再用副本计算。

如何: var date2=new Date(date1)

原理: 日期对象中保存的其实是1970年1月1日0点至今的毫秒数

为什么: 用字符串保存时间，2大缺点:

1. 时区的误差

2. 不便于算术计算

其实: 用时间段表示时间，好处:

1. 时间段，在各个时区是同步的，没有误差的

2. 时间段便于算术计算

结论: 在程序中，存储时间或操作时间，都用date

要在数据库中存储时间或传输时间数据，都用整数（长整数）

问题: 人看不懂毫秒数

解决:

API: 3句话

1. 单位: FullYear Month Date Day

Hours Minutes Seconds Milliseconds

2. 每个单位都有一对儿get/set函数

其中: date.get单位() 获得指定单位上的数值

date.set单位(值) 修改指定单位的数值

福利: set可自动计算进制

特例: Day没有set函数，不能修改

3. 取值范围:

只有Date从1~31，其余都是从0~进制-1

Month: 0~11 计算机中的月份比现实中数值-1

Date: 1~31

Day: 0~6

Hours: 0~23

Minutes/Seconds: 0~59

日期计算: 2种:

1. 两个日期对象可直接相减: 结果是毫秒差

2. 对任意单位做加减

3步:

1. 取值: var d=now.getDate(); //17

2. 做计算: d+=30; //47

3. 放回去: now.setDate(d)

可简写为一步:   
 now.setDate(now.getDate()+30)

问题: setDate直接修改原日期对象

解决: 如果希望同时保留计算前后的时间时，都要先复制开始时间的副本，再用副本计算结束时间

日期格式化:

date.toString() 转为当地时间的完整日期格式

date.toLocaleString() 转为当地时间的简版日期格式

date.toLocaleDateString() 转为当地时间的简版格式，仅包含日期部分

date.toLocaleTimeString() 转为当地时间的简版格式，仅包含时间部分

date.toGMTString() 转为0时区标准时间

自定义:

2. Error:

什么是: 保存错误信息的对象

错误处理: 即使程序出错，也保证程序不中断退出的一种机制。

为什么: 程序只要出错，就会中断退出

何时: 只要希望即使程序出错，也不中断退出。

如何:

try{

可能出错的正确的语句

}catch(err){

只有出错时，才执行的错误处理代码:

1. 提示用户; 2. 记录日志; 3. 保存数据

}

问题: 1. 效率: try catch效率比普通程序低

2. 无法精确预报错误

3. 自动创建错误对象，占用更多内存空间

解决: 用if...else提前判断可能出错的原因

鄙视: js中错误对象有几种类型: 6种:

SyntaxError 语法错误

ReferenceError 引用错误——要用的变量没找到

TypeError 类型错误——错误的调用了函数或对象的API

RangeError 范围错误——参数值超范围

URIError URI错误——违反URI标准

EvalError Eval错误

主动抛出异常:

throw new Error("错误信息")

何时: 在协作开发中，程序的作者，向程序的调用者抛出错误，告知调用者错误的使用了程序。

3. Function

什么是: 保存一项任务步骤清单的代码段

为什么: 重用一项任务的步骤清单

何时: 只要程序中出现重复的步骤，都要先封装在一个函数中。再反复调用函数。

如何:

创建函数对象: 3种:

1. 声明方式:

function 函数名/任务名(参数变量列表){

步骤清单;

return 结果;

}

参数变量: 接收函数运行时必须数据的变量

何时: 如果函数步骤中必须某些数据才能正常执行

强调: 参数也是局部变量

返回值: 函数的执行结果

何时: 如果调用者需要获得函数的执行结果时

缺点: 会被声明提前

优点: 无论函数定义在哪儿，都可随时调用

解决:

2. 赋值:

var函数名/任务名=function(参数变量列表){

步骤清单;

return 结果;

}

优: 避免声明提前!

缺: 必须在调用前先定义函数，才能调用

3. 用new:

var 函数名=new Function(

"参数1",

"参数2",

...,

"函数体; return 返回值"

);

揭示了: 函数其实也是一个对象

函数名其实只是一个普通变量, 只是引用函数对象的地址而已。

鄙视题:

以下js函数定义正确的是:

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

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

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

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

调用函数:

var 变量=函数名(参数值列表);

重载(overload):

什么是: 相同函数名不同参数列表的多个函数，在调用时可根据传入参数的不同，自动选择匹配的函数调用。

何时: 只要一项任务，可能根据传入参数的不同，执行不同的流程时。

为什么: 减少API的数量，减轻调用者的负担

问题: js语法默认不支持重载

js不允许多个同名函数同时存在!

解决: arguments

什么是: 每个函数中，默认自动创建的，能接收所有参数值的类数组对象(长得像数组的对象)

vs 数组:

相同: 1. 下标; 2. .length; 3.遍历

不同点: 类型不同——API不通用

匿名函数:

什么是: 定义函数时不指定函数名的函数

何时: 只执行一次的函数都不用起名

为什么: 节约内存——用完后立刻就能释放!

如何: 2种:

1. 回调: 将一个函数传递给另一个函数去调用

何时: 只要一个函数，只执行一次时

比如: 定时器 setInterval(fun,ms)

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

str.replace(/正则/ig,function(kw){...})

2. 自调: 创建一个函数后，立刻调用自己!

何时: 划分临时作用域，避免使用全局变量

为什么: 全局变量会驻留内存，造成污染

如何: (function(){  
 ...  
 })();

总结: 今后，所有页面功能，都必须放在一个匿名函数中执行.

补: 垃圾回收:

什么是: 引擎可自动释放不再被任何变量使用的对象的内存.

什么是垃圾回收器: 专门监控内存中对象的引用次数，并释放不再被使用的对象 的小程序

特点: 伴随主程序启动而启动，不会干扰主程序的执行  
 如何:

1. 垃圾回收器伴随主程序运行而并行执行

但不会干扰主程序

2. 垃圾回收器记录每个对象被几个变量引用着

3. 如果对象还有变量引用，则不回收

如果对象不再被任何变量引用，则自动回收

我们做什么: 每当用完一个大对象，都要尽快将变量赋值为null，释放对对象的引用。