张东

[zhangdong@tedu.cn](mailto:zhangdong@tedu.cn)

准备:

1. 犀牛书:

2. 微信公众号: 前端大全

3. 上届笔记: 打印 预习

谁抄笔记抄的越勤，谁学的越烂！

正课:

1. String

2. 正则表达式

1. String

什么是: 由多个字符组成的字符数组

vs数组: 相同: 1. 下标, 2. .length, 3. for遍历

4. .slice(starti,endi+1)

不同: 类型不同, API不通用!

API: 所有字符串API都无权修改原字符串，只能返回新字符串

大小写转换:

str.toUpperCase()

str.toLowerCase()

何时: 不区分大小写时

获取指定位置的字符:

str[i] str.charAt(i)

获取指定位置的字符的unicode号

var code=str.charCodeAt(i)

反向将unicode号转回字:

var char=String.fromCharCode(code)

选取子字符串:

str.slice(starti,endi+1)

str.substring(starti,endi+1) 唯独不支持负数参数

str.substr(starti,n) 从starti开始，选n个！

查找关键词: 4种:

1. 查找一个固定的关键词出现的位置:

var i=str.indexOf(“关键词”[,fromi])

如果找不到，返回-1

var i=str.lastIndexOf(“关键词”)

问题: 只能查找一个固定的关键词，只要关键词发生变化，就找不到了！

2. 用正则判断是否包含关键词:

var i=str.search(/reg/);

查找str中第一个符合reg规则的关键词的位置i

如果找不到，返回-1

2. 正则表达式(regular Expression):

什么是: 规定字符串中字符出现规律的规则

何时: 2个:

1. 按规则模糊查找多种关键词

2. 验证字符串格式

如何:

1. 关键词原文，就是最简单的正则！

2. 字符集:

什么是: 规定字符串中一位字符上备选字符列表的集合

何时: 只要关键词中，一位字符上有多种备选字时

如何: [备选字列表]

简写: 如果备选字列表中的字符是连续的

可用-省略中间字符

一位数字: [0-9]

一位大写字母: [A-Z]

一位小写字母: [a-z]

一位字母: [A-Za-z]

一位数字或字母: [A-Za-z0-9]

一位汉字: [\u4e00-\u9fa5]

反选: 不包含: [^47]

3. 预定义字符集:

对常用字符集的更简化写法

\d 一位数字 [0-9]

\w 一位字母，数字或下划线[A-Za-z0-9\_]

\s 一位空字符: 空格，Tab…

. 通配符 可代表一切字符

问题: 无法灵活规定字的数量

解决: 量词

4. 量词:

什么是: 规定一位字符出现次数的规则

何时: 只要规定一位字符出现的次数

如何: 字符集量词

包括: 2大类:

1. 有明确数量边界的:

{6,8} 6到8位

{6,} 6位以上

{6} 必须6位

比如: 车牌号: [\u4e00-\u9fa5][A-Z]·[A-Z0-9]{5}

2. 没有明确边界的:

\* 可有可无，数量不限

? 可有可无，最多1个

+ 至少一个，多了不限

5. 选择和分组:

选择: 或 多个规则匹配其一即可

规则1|规则2|…

分组: () 将多个规则，包裹为一组

何时: 只要希望一个量词同时修饰多个字符集时，都要先将字符集用()包裹为一组，再用量词修饰整个分组!

比如: 身份证号:

15位数字 2位数字 1位数字或X

\d{15} ( \d\d [0-9X] )?

后三位整体可有可无，最多1次

比如: 手机号:

+86或0086 可有可无，最多1个

空字符 可有可无，多了不限

1

在3456789 挑一个

9位数字

(\+86|0086)?\s\*1[3-9]\d{9}

6. 匹配位置:

^ 字符串的开头

$ 字符串的结尾

何时: 仅匹配字符串开头或结尾的内容时

比如: 开头的空字符 ^\s+

结尾的空字符 \s+$

开头和结尾的空字符 ^\s+|\s+$

\b 单词边界: ^ $ 空字符 标点

比如: \bno\b 选择单词no

\b[a-z] 单词首字母

作业:

微信 微 信 weixin wei xin wx w x wechat

正课:

1. String  
2. RegExp

3. Math

4. Date

1. String:

查找关键词: 4种

1. 查找一个固定的关键词的位置:

var i=str.indexOf(“关键词”,fromi)

var i=str.lastIndexOf(“关键词”)

问题: 只能查找一个固定的关键词。如果关键词发生变化，就找不到了

解决: 正则!

2. 用正则表达式判断是否包含敏感词:

var i=str.search(/正则/)

如果i不是-1，说明包含

如果i是-1，说明不包含

问题: 正则表达式天生区分大小写！

解决: 在第二个/后加后缀i, 表示ignore

何时: 只要查找英文/拼音时，都要加i

问题: 只能找一个，且只能获得位置，不能获得内容

3. str.match() 2种:

1. 查找第一个关键词的内容和位置

var arr=str.match(/正则/i)

返回值: arr:[ 0: 关键词, index: 位置 ]

找不到, 返回null!

2. 查找所有关键词的内容

var arr=str.match(/正则/ig) g-global

返回值: arr [“关键词1”,”关键词2”,…]

问题: 只能查找内容，无法查找位置

4. 即找每个关键词内容，又找位置

?

替换: 2种:

1. 简单替换: 将所有敏感词替换为统一的内容

2种:

1. 关键词固定:

str=str.replace(“关键词”,”替换值”);

强调: 无权直接替换原字符串，只能返回新字符串，必须用变量接住替换后的新值

问题: 只能替换一个关键词!

2. 关键词变化:

str=str.replace(/正则/ig,”替换值”)

2. 高级替换: 根据不同的敏感词，动态选择替换不同的值

str=str.replace(/正则/ig,function(kw){

//kw会自动接收到，当前找到的一个关键词

return 根据kw的不同，返回不同的值

})

衍生: 删除关键词: 将关键词替换为””

切割:

1. 简单切割: 按照固定的分隔符将字符串切割为多段子字符串

var subs=str.split(“切割符”)

强调: 在结果数组中，不包含切割符

固定用法: 将字符串打散为字符数组

var arr=str.split(“”);

2. 复杂切割: 切割符不确定的情况

var subs=str.split(/正则/)

2. RegExp

什么是: 专门保存一条正则表达式的对象，并提供了用正则表达式执行验证和查找的API

何时: 只要在程序中保存一条正则表达式，都要用正则表达式对象

如何: 2种:

1. 直接量: var reg=/正则/ig

何时: 只要正则表达式是固定不变的

2. 用new: var reg=new RegExp(“正则表达式”,”ig”)

何时: 如果正则表达式需要用程序动态生成

API: 2个:

1. 验证格式:

var bool=reg.test(str);

检查字符串str的格式，是否符合正则reg的要求

问题: 默认，只要字符串中部分规则和正则相符，就返回true，不要求完全匹配

解决: 前加^后加$, 意为从头到尾必须完整匹配

2. 即找所有关键词内容，又找位置！

var arr=reg.exec(str)

在str中查找下一个符合reg要求的关键词的内容和位置

返回值: arr [ 0: 关键词, index: 位置]

如果找不到，返回null !

3. Math:

什么是: 专门封装数学计算的常量和API的对象

何时: 只要使用数学计算的常量或进行数学计算时

如何: 强调: 不能new！所有属性和API都用大写的Math直接调用

API:

1. 取整: 3种:

1. 上取整: Math.ceil(num)

2. 下取整:

Math.floor(num) 只能处理纯数字

parseInt(str) 可去掉数字后的非数字字符

3. 四舍五入取整:

Math.round(num) 优: 返回数字，可直接做计算

缺: 只能取整，不能规定小数位数

num.toFixed(d) 缺: 返回字符串

优: 可灵活规定小数位数

2. 乘方和开平方

Math.pow(底数,幂)

Math.sqrt(num)

3. 最大值，最小值:

Math.max(值1,值2,…);

Math.min(值1,值2,…)

问题: 不支持从数组中获得最大值和最小值

解决: Math.max(…arr)

…可将数组打散为单个值，再传递给函数

4. 三角函数:

Math.atan(对边长/临边长)

问题: 无法区分象限

解决: 今后算角度，一律用atan2

Math.atan2(对边长, 临边长)

5. 随机数:

Math.random() 在0~1之间生成随机小数

在min~max之间取随机:

parseInt(Math.random()\*(max-min+1)+min)

正课:

1. Date

2. Error

3. Function

1. Date:

什么是: 存储一个时间，并提供操作时间的API

何时: 只要在程序中存储或计算时间都用日期对象

如何:

创建: 4种:

1. 创建日期对象，并自动获得当前系统时间:

var now=new Date();

强调: 因为js是在客户端执行，所以new Date()只能获得客户端系统时间。

2. 创建日期对象，并保存自定义时间:

var date=new Date(“yyyy/mm/dd hh:MM:ss”);

3. 复制一个日期对象:

var date2=new Date(date1)

4. 将毫秒数，转化为当地时间:

var date=new Date(ms)

何时: 将数据库中保存的毫秒数，转化为人能看懂的日期对象，且换算成当地时间。

本质: 计算机中日期对象中保存的不是字符串，而是一个毫秒数。1970年1月1日（0时区国际标准时间）至今的毫秒数

字符串的缺点: 1. 受时区影响； 2. 不便于计算

毫秒数的优点: 1. 不受时区影响；2. 极其便于计算

缺点: 人看不懂！

所以，今后数据库中保存时间都用毫秒数

API: 3句话:

1. 8个单位:

FullYear Month Date Day

Hours Minutes Seconds Milliseconds

2. 每个单位上都有一对儿get/set方法

其中: getXXX() 专门用于获取单位的值

setXXX(num) 专门用于设置单位的值

强调: setXXX(num) 可自动计算时间进制

Day没有set, 因为星期几不能随便改

3. 取值范围:

Month: 0~11 计算机中的月份都比现实中小1

Date: 1~31

Day: 0~6

Hours: 0~23

Minutes/Seconds: 0~59

计算: 2种:

1. 两日期对象可相减: 结果是毫秒差

2. 对任意单位做加减: 3步:

1. 取值: var d=now.getDate(); //22

2. 计算: d+=60 //82

3. set回去: now.setDate(d); //setXXX可自动调整进制

强调: setXXX()直接修改原日期对象

如果想同时保存计算前后的两个日期对象，应该在计算之前复制原日期对象的副本，再用副本计算新日期

简写: date.setXXX(date.getXXX()+n)

格式化:

.toString() 转为当地时间的完整格式

.toLocaleString() 转为当地时间的简化格式

.toLocaleDateString() 当地时间的简化格式，仅保留日期部分

.toLocaleTimeString() 当地时间的简化格式，仅保留时分秒部分

.toGMTString() 转为国际标准时间(0时区)

2. Error:

什么是: 保存错误信息的对象

如何:

创建: 在发生错误时，自动创建

包括: 错误的类型，错误的原因和出错的位置

问题: 所有程序，一旦发生错误，就会强行退出

解决:

错误处理: 即使发生错误，也保证程序不中断退出的一种机制。

何时: 几乎所有的程序，只要可能发生错误，都要提前进行错误处理

如何:

try{

可能发生错误的正确代码

}catch(err){

只有发生错误时，才执行的错误处理代码:

包括: 提示用户，保存数据，记录日志

}

问题: 1. 被动的接受错误，无法精确提示用户

2. try catch比正常程序执行效率低

解决: 尽量提前预知错误原因，用if else 代替try catch

鄙视: js中错误的类型有几种:

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: 错误的使用了类型

RangeError 范围错误: 参数超范围

URIError EvalError

抛出自定义错误:

throw new Error(“自定义错误消息”)

何时: 在协作开发中，只要函数的作者，向函数调用者抛出错误，说明错误的使用了函数

3. Function

什么是: 内存中保存一段代码段的对象

何时: 当程序中，有一项任务可能被反复使用时，都要将任务的代码段，先保存在函数对象中，再反复调用函数执行

为什么: 复用

如何:

创建: 3种

1. 直接量:

function 函数名(参数列表){

代码片段

return 返回值

}

问题: 被声明提前(hoist)!

解决: 用赋值方式创建函数对象

2. 赋值:

var 函数名=function (参数列表){

代码片段

return 返回值

}

优点: 不会被声明提前

揭示了: 函数名，其实也是一个普通的变量

函数其实是一个对象

函数名变量通过地址引用函数对象

3. 用new:

var fun=new Function(“a”,”b”,…,”函数体; return xxx;”);

鄙视: 以下创建函数正确的是:

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，在调用时可根据传入参数的不同，自动选择对应的函数调用。

为什么: 减少API的数量，减轻调用者的负担

何时: 如果一件事，根据传入的参数不同，执行不同的操作时

如何:

问题: js默认不支持重载！

为什么: js不允许多个同名函数同时存在

解决: 用arguments对象，变通实现:

arguments: 每个函数内自动创建的接收所有传入函数的参数的对象

arguments是类数组对象: 长得像数组的对象

vs 数组: 相同: 1. 下标, 2. Length 3. for遍历

不同: 类型不同，API不通用!

正课:

1. 垃圾回收

2. Function  
3. \*\*\*OOP

1. 垃圾回收(GC):

什么是: 引擎会自动检测不再使用的对象，并自动释放其内容空间

为什么: 内存容量是极其有限的

何时: 只要引擎检测到一个对象不再被任何变量使用，就会自动释放该对象的内容空间

垃圾回收器: 专门检测并回收垃圾对象的小程序

如何:

1. 垃圾回收器伴随主程序执行而并行执行

2. 垃圾回收器会记录每个对象被几个变量引用着

3. 只要垃圾回收器发现一个对象不再被任何变量引用，就自动释放该对象。

强烈建议: 如果一个对象不再被使用，应尽在将变量赋值为null，释放对该对象的引用。

2. Function

匿名函数:

什么是: 定义时未指定函数名的函数

何时: 只要一个函数只用一次，不会重复使用时

为什么: 节约内存

如何: 2种:

1. 回调函数: 2种:

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

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

2. 匿名函数自调:

什么是: 临时创建一个匿名函数后，立刻调用自己

为什么: 避免使用全局变量,造成全局污染

何时: 所有功能代码都要放在匿名函数自调内

如何: (function(参数列表){

函数体;  
 return 返回值

})(参数值列表)

\*\*\*作用域和作用域链:

作用域(scope):

什么是: 一个变量的使用范围

为什么: 为了避免函数内外的变量间互相干扰

包括: 2种:

1. 全局作用域: 保存着所有全局变量/函数

2. 函数作用域: 保存着函数内的局部变量

变量的使用顺序: 先用局部变量，局部没有，才去全局找

函数的生命周期:

1. 程序执行前:

创建一个数组(执行环境栈ECS): 用于记录正在执行的函数

浏览器本身也是一个程序, 执行环境栈中默认首先记录的是浏览器的主程序main()

main()函数执行时，会创建全局作用域对象(window)，保存所有浏览器内置的对象和方法

2. 函数定义时:

在全局创建函数名变量

在window之外创建函数对象保存函数定义

函数名变量通过地址引用函数对象

函数对象使用scope属性指回自己来自的作用域

3. 函数调用时:

先再ECS中添加本次函数调用的记录

为本次函数调用创建专门的函数作用域对象(AO)

在函数作用域对象中保存本次函数调用所需的所有局部变量

函数作用域对象的parent属性指向函数来自的父级作用域对象

变量的使用顺序: 就近

先用局部的，如果局部没有才去全局window中找

4. 函数调用后:

将本次函数调用的记录从ECS中出栈

导致函数作用域对象释放

导致局部变量一同释放

所以: 局部变量不可重用！

作用域链: 由多级作用域逐级引用形成的链式结构

保存了：所有的变量

控制了：变量的使用顺序: 先局部，后全局

\*\*\*\*\*闭包(closure):

什么是: 即重用变量，又保护变量不被篡改的一种机制

为什么:

全局变量: 优: 可重用; 缺: 随处可用,易被篡改

局部变量: 缺: 不可重用; 优: 仅函数内可用，不会被篡改

何时: 即重用变量，又保护变量不被篡改

如何: 3步: 判断闭包的三个特点:

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对象返回到外部: 3种途径

1. 用return

2. 直接给全局变量赋值

3. 将内层函数包裹在数组/对象中放回

3. 使用者调用外层函数，获得内层函数的对象

闭包是怎么形成的: 外层函数的函数作用域对象(AO)无法释放

鄙视: 2步:

1. 找受保护的变量: 外层函数的局部变量

2. 找外层函数共返回哪些内层函数: 一次外层函数调用，返回的多个内层函数，公用同一个闭包中的受保护的变量

闭包的问题: 内层函数比普通函数占用更多内存空间(外层函数的AO)

解决: 一旦闭包不再使用，应立刻释放闭包结构：

getNum=null

3. \*\*\*OOP

什么是: 程序中，都是用对象来集中描述现实中一个具体事物的属性和功能。

为什么: 为了便于大量数据和功能的使用和维护

何时: 今后几乎所有的程序，都采用面向对象的方式

如何: 3大特点: 封装 继承 多态

封装:

什么是: 创建一个对象，集中保存一个事物的属性和功能

为什么: 便于查找和维护

何时: 只要使用面向对象编程，都要先创建对象

如何:

事物的属性会成为对象的属性

事物的功能会成为对象的方法

成员=属性+方法

3种:

1. 对象直接量:

var obj={

属性名:值,

属性名:值,

方法名:function(){

… this.属性名 …

},

方法名:function(){

… this.属性名 …

}

}

访问对象的成员:

访问对象的属性: 对象名.属性名

本质: 属性其实是保存在对象中的变量

访问对象的方法: 对象名.方法名()

本质: 方法其实是保存在对象内的函数

问题: 对象自己的方法要使用自己的属性

错误1: 在方法中直接使用属性名

为什么: 默认, 没有加任何前缀的普通变量，只能在作用域链中查找，无权进入任何对象查找

错误2: 在方法中使用 对象名.属性名

为什么: 对象名只是一个普通的变量，随时可能发生变化

正确做法: 只要方法想访问对象自己的属性，必须用this.属性名

this->自动获得正在调用方法的.前的对象名

2. 用new创建:

var obj=new Object();  
 obj.属性名=值;

obj.方法名=function(){

… this.属性名 …

}

本质: js中一切对象底层都是关联数组:

vs 关联数组:

相同: 1. 可用["下标"], 也可用.下标方式访问成员

如何选择:

1. 如果属性名是固定的，首选.

2. 如果属性名是动态获得的，应用[]

2. 随时可在任意位置添加新成员

3. 访问不存在的成员不会报错，返回undefined

正课:

1. \*\*\*OOP

1. \*\*\*OOP

封装: 3种:

1. 对象直接量:

2. 用new

问题: 一次只能创建一个对象，如果反复创建多个对象，代码会很冗余

解决: 用构造函数反复创建多个结构功能相同的对象

3. 用构造函数

什么是: 描述一类对象统一结构的函数

何时: 只要反复创建相同结构的多个对象时

为什么: 代码重用

如何:

定义构造函数:

function 类型名(属性参数,…){

this.属性名=属性参数;

… = …;

this.方法名=function(){

… this.属性名 …

}

}

用构造函数创建对象:

var obj=new 类型名(属性值,…)

new: 4件事:

1. 创建一个空对象

2. 让新的子对象继承构造函数的原型对象

3. 调用构造函数:

1. this->new

2. this.属性名=值: 通过强行赋值的方式，为对象添加新属性，并保存属性值

4. 返回新对象的地址，保存在变量中

问题: 每个对象都有一个重复的相同的方法定义，浪费内存

解决: 继承

继承:

什么是: 父对象的成员，子对象无需重复创建，即可直接使用

为什么: 代码重用, 节约内存

何时: 只要多个子对象，拥有相同的属性值和方法时

如何: 通过原型对象

什么是原型对象: 集中保存子对象共有成员的父对象

如何:

创建: 不用创建:

定义构造函数时，自动附赠一个原型对象

创建子对象时: new的第2步，让子对象自动继承构造函数的原型对象

添加共有成员:

构造函数中不再包含共有方法的定义

所有共有的方法都应添加到构造函数的原型对象中

构造函数.prototype.方法名=function(){

… this.属性名 …

}

自有属性和共有属性:

自有属性: 保存在子对象本地，归子对象独有的属性

共有属性: 保存在父对象中，所有子对象共享的属性

获取属性值: 子对象.属性名 ——相同

修改属性值: 自有属性只能用子对象修改

共有属性只能用原型对象修改

内置类型的原型对象:

鄙视: ES中内置类型/对象: 11个

String Number Boolean ——包装类型

Array Date RegExp Math

Error

Function Object

Global(在浏览器中被window代替)

本质:

所有能new的，都是构造函数: Array, Date, RegExp, …

所有的API，都保存在构造函数的原型对象中:

Array.prototype Date.prototype

问题: 旧浏览器无法使用新的API

解决: 让旧浏览器也支持新的API

如何: 向该类型的原型对象中添加自定义的API

开发时: MDN 类型名.prototype.函数名->Polyfill

鄙视: 包装类型:

什么是: 专门保存一个原始类型的值，并提供操作原始类型值的API

为什么: 原始类型的一个值本身，没有任何功能

何时: 只要试图对原始类型的值调用API时，都会自动使用该原始类型对应的包装类型对象

如何:

创建: 不用自己创建

当试图对原始类型的值调用API时，自动创建对应的包装类型对象。

调用的API其实是包装类型对象提前定义好的API

问题: 包装类型的API，也有浏览器兼容性问题

解决: 同内置类型的API解决方法一样

原型链:

什么是: 由多级父对象逐级继承，形成的链式结构

包含了: 所有对象的属性和方法

控制着: 属性和方法的使用顺序，以及共享范围

鄙视: 判断一个对象是不是数组类型，有几种方法:

问题: typeof只能区分五种原始类型和function

无法进一步细致区分引用类型对象的类型

1. 判断原型对象:

var bool=Array.prototype.isPrototypeOf(obj)

判断Array.prototype是不是obj的父级原型对象

2. 判断构造函数:

//obj.constructor===Array

var bool= obj instanceof Array

问题: 不仅检查直接父级，且检查整个原型链

检查不够严格

3. 判断每个对象内部的隐藏属性class:

class属性在创建对象时保存对象的类型

不随继承关系改变而改变

如何: 唯一的办法: Object.prototype.toString

问题: 多数父对象都重写了toString方法，将顶级的toString覆盖了，无法直接调用到

解决: call: 让任何一个对象抢到任何一个想调用的函数执行

要调用的函数.call(obj)

多态:

什么是: 同一个函数，在不同情况下表现出不同的状态

包括: 2种:

1. 重载:

2. 重写(override):

什么是: 子对象中定义了和父对象中的API同名的成员

为什么: 子对象觉得父对象中的成员不好用

何时: 只要子对象觉得父对象中的成员不好用，就可以在子对象本地重写父对象成员

如何: 只要在子对象本地定义和父对象中同名的API

正课:

1. \*\*\*OOP

2. \*\*\*ES5

1. \*\*\*OOP

静态方法: static

什么是: 无需创建子对象，就可用构造函数直接调用的方法。

为什么: 有些方法不确定使用它的对象的类型，所以不能放在原型对象内部

何时: 只要一个方法，不确定使用它的对象的类型时，任意对象都可使用时，就用静态方法。

如何: 将方法直接定义在构造函数上

自定义继承:

1. 仅修改一个对象的父对象:

//child.\_\_proto\_\_=father

Object.setPrototypeOf(child,father)

2. 修改所有子对象的父对象:

构造函数.prototype=father

强调: 在创建子对象之前

3. 两种类型间的继承:

何时: 如果发现多个类型之间拥有部分相同的属性结构和方法定义时

如何: 2大步:

1. 定义抽象父类型: 2件事:

1. 在构造函数中定义相同的部分属性

2. 在原型对象中定义相同的方法

2. 让子类型继承父类型: 2件事:

1. 让子类型构造函数借用父类型构造函数

错误: 直接调用父类型构造函数:

结果: 父类型构造函数中的属性，全部泄露到全局

为什么: 一个函数，不用new也不用.调用，this默认都指window

正确: 今后只要函数中的this不是想要的，可用call随意更换为想要的对象

函数.call(正确的对象,其他参数值,....)

2. 让子类型原型对象继承父类型原型对象

2. \*\*\*ES5

严格模式:

什么是: 比普通的js运行机制要求更严格的运行模式

为什么: 解决js中部分广受诟病的缺陷

何时: 几乎所有代码都要运行在严格模式下

如何: 2种情况:

1. 新项目: 在所有js文件/script标签的顶部添加:

"use strict";

2. 旧项目升级: 逐个函数向严格模式迁移:

在函数定义内的顶部添加"use strict";

要求:

1. 禁止给未声明的变量赋值

针对缺陷: 给未声明的变量赋值，会自动在全局创建该变量，造成内存泄露

禁止后: 避免了因为全局变量造成的内存泄露

2. 将静默失败升级为错误

什么是静默失败: 运行即不成功，又不报错！

3. 所有普通函数调用和匿名函数中的this不再默认指向window，而是undefined

避免内存泄露

4. 不再推荐使用arguments.callee

什么是arguments.callee: 可在函数调动时，自动获得函数本身

其实，是禁用递归。因为递归效率极低，因为重复计算量太大！

解决: 几乎所有递归，都能用循环代替

保护对象: 2个层面

保护属性:

为什么: 普通js中对象的属性值可以是任何值，可以随意修改。

如何保护:

ES5将对象的属性分为2大类:

命名属性: 可用.直接访问到的属性

内部属性: 不能用.直接访问到的属性

\_\_proto\_\_ constructor class

如何保护命名属性:

ES5将命名属性又分为2大类:

数据属性: 直接保存属性值的属性

如何保护数据属性: 4大特性:

每个数据属性都又包含四个特性:

value: 值 //实际存储属性值

writable: bool //控制当前属性是否可修改

enumerable: bool //控制当前属性能否被for in遍历到

configurable: bool //控制能否修改其它特性

//控制能否删除该属性

强调: enumerable只能控制for in变量。用.依然可访问到

强调: configurable一旦改为false，不可逆！

所以，经常作为其它两个特定的双保险同时关闭

获取一个属性的四大特性:

var attrs=Object.getOwnPropertyDescriptor(obj,"属性名")

修改一个属性的四大特性:

Object.defineProperty(obj,"属性名",{

特性:值, 特性:值, ...

})

问题: 一次只能修改一个属性的四大特性

解决:

Object.defineProperties(obj,{

属性名:{

特性:值, 特性:值,...

},

属性名:{

特性:值, 特性:值,...

},

... ...

})

数据属性的问题: 无法用自定义规则保护属性

解决: 用访问器属性保护数据属性

访问器属性: 不实际存储属性值，仅提供对其他数据属性的保护

如何定义: 不能用直接量定义，只能用defineProperty/defineProperties方法添加

Object.defineProperty(obj,"访问器属性名",{

//四大特性:

get:function(){ return this.数据属性名},

set:function(val){

if(val符合要求)

this.数据属性=val;

else

throw new Error("错误提示！")

},

enumerable:bool,

configurable:bool

})

如何使用访问器属性:

不用自己调用get和set

访问器属性的用法和普通数据属性完全一样！

原理: 当试图获取属性值时，其实自动调动get

当试图修改属性值时，自动调用set方法，且自动将新值传给set的参数val

问题: 如果用普通的数据属性，保存访问器属性要保护的值，则可能绕过访问器属性，直接修改数据属性的值

解决: 将受保护的属性值，保存在闭包中

如何:

保护对象结构:

正课:

1. \*\*\*ES5

2. ES6

1. \*\*\*ES5

保护对象:

保护属性:

命名属性:

数据属性:

四大特性: value, writable, enumerable,configurable

访问器属性:

四大特性: get(), set(val), enumerable,configurable

构造函数中定义保护:

内部属性:

保护结构: 3个层次

1. 防扩展: 禁止向对象中添加新属性

Object.preventExtensions(obj)

原理: 每个对象内，都有一个内部的隐藏属性:

extensible:true

preventExtensions将obj内的extensible改为false

2. 密封: 防扩展的同时，进一步禁止删除现有属性

Object.seal(obj)

原理: 1. extensible改为false

2. 自动将每个属性的configurable都改为false

3. 冻结: 密封的同时，禁止修改属性值

Object.freeze(obj)

原理: 1. extensible改为false

2. 自动将每个属性的configurable都改为false

3. 自动将所有属性的writable改为false

Object.create()

什么是: 创建一个新对象，继承现有的一个指定父对象，同时还可为新对象添加自有属性

何时: 如果没有构造函数，只有父对象，也想创建子对象时

如何:

1. 简单: 仅继承:

var child=Object.create(father)

2. 高级: 即继承，又扩展自有属性:

var child=Object.create(father,{

//defineProperties的格式一样的

属性:{  
 特性:值,

... : ...  
 },

属性:{

特性:值,

... : ...

}

})

call(),apply(),bind()

共同点: 都是替换函数中不想要的this为想要的对象

何时: 只要一个函数调用时，其中的this不是想要的就可更换

如何:

1. call/apply

强行调用指定函数，并临时替换函数中的this为指定对象

强调: 立刻执行，且仅执行一次

执行时，临时替换时，执行后this回复原样

如何: 指定函数.call(替换this的对象, 参数值,.....)

call vs apply: 参数值列表

call: 要求传入函数的参数值列表，必须分别传入

apply: 要求传入函数的参数值列表，必须放在数组中整体传入.

2. bind()

什么是: 创建一个和原函数完全相同的函数，并永久绑定函数中的this为指定对象

如何: var 新函数=原函数.bind(替换this的对象,参数值,...)

基于原函数，创建一个新函数，并永久绑定this为指定对象

数组API:

//indexOf

判断:

every(): 判断数组中是否所有元素都符合要求

some(): 判断数组中是否包含符合条件的元素

如何:

var bool=arr.every(function(elem,i,arr){

//elem: 当前元素值

//i: 当前位置

//arr: 当前数组对象  
 return 判断条件  
 });

运行过程: every会拿着回调函数在每个元素上执行一次，只要碰到不满足条件的，就退出，并返回false。如果走到头了，就返回true。

遍历API:

forEach(): 对原数组中每个元素，执行相同操作

map(): 取出原数组中的元素值，执行相同操作后，放入新数组中返回。原数组保持不变。

如何:

arr.forEach(function(elem,i,arr){

arr[i]=新值

})

var newArr=arr.map(function(elem,i,arr){

return 新值

})

过滤和汇总:

过滤: 选取出原数组中符合条件的元素，组成新数组。原数组保持不变。

如何: var sub=arr.filter(function(elem,i,arr){

return 判断条件

})

只有符合条件的元素，才会放入新数组sub中

汇总: 将数组中每个元素值，汇总计算出一个结果

如何: var result=arr.reduce(function(prev,elem,i,arr){

//prev 截止目前的元素的临时汇总值

return prev+elem;

}, base)

2. ES6

不改变原理的基础上，尽量简化开发

let: 代替var，用于声明变量

为什么: 2个问题: 声明提前, 没有块级作用域

要求:

1. 不会被声明提前:

检查，不允许提前使用未声明的变量

2. 添加块级作用域:

什么是: if while do while for 都是一级作用域

为什么: 避免块内的变量影响外部

let声明的变量，只在当前块内有效

原理:

其实是匿名函数自调！

let+for: 闭包的效果

参数增强:

默认值: 定义函数时，可提前为部分参数变量指定默认值

如何: function fun(参数1, 参数2, ... , 参数n=默认值, 参数n=默认值)

强调: 有默认值的参数必须放在参数列表的结尾

剩余参数: 代替arguments

arguments的问题: 2个

1. 不是数组:

2. 必须完整获得所有参数:

如何: 定义时: function fun(参数1,参数2,..., ...数组名)

除了前边明确的几个参数值外，剩余参数，都收集到数组中保存

优点: 1. 是纯正的数组类型

2. 可随意选择开始获取参数值的位置

正课:

1. ES6:

1. ES6:

参数增强:

默认值default:

剩余参数rest:

散播spread: 将一个数组打散后，分别传入函数中

何时: 如果一个函数需要多个参数，而给定的值缺来自一个数组时。

如何: fun(...数组)

原理: fun.apply(null, 数组)

箭头函数: 简化一切回调函数/匿名函数自调

如何: 3句话:

1. 所有匿名函数(回调/自调)都可去function，加=>

2. 如果参数只有一个，可继续省略()

3. 如果函数体只有一句话，可省略{}

如果唯一的一句话是return，可省略return

特点: 内外共用同一个this!

如果不希望内外this通用，就不应用箭头函数简化

模板字符串: 代替+拼接字符串

什么是: 为动态生成打断内容，而提前定义的字符串模板

何时: 只要字符串拼接，一律使用模板字符串

如何:

1. 用反引号``包裹字符串模板

优点: 支持换行，且不会和""或''冲突

2. 如果动态生成的内容/js表达式，必须用${}包裹

解构:

什么是: 将一个对象/数组，拆散到多个变量中

何时: 只要在一个大的对象/数组中选取部分属性/元素放入变量中使用时

如何: 3种:

1. 数组解构: 下标对下标

var date=[2018,2,28];

0 1 2

var [y,m,d]=date;

0 1 2

y=2018, m=2, d=28

2. 对象解构: 属性对属性

var dbconfig={  
 ip:"xxx",  
 port:"xxx",  
 db:"xxx",  
 user:"root",  
 pwd:"123456"  
 }

var {ip:ip,user:user,pwd:pwd}=dbconfig;

ip user pwd

3. 参数解构: 原理同对象解构

问题: 传统参数定义是固定个数，固定先后顺序，无法灵活选择要传入的参数和不要传入的参数

定义函数时: function fun(ip,port,db,user,pwd)

解决:

定义函数时:

function fun({ip,port,db,user,pwd })

调用时:

fun({ip:"xxx",db:"xz"})

fun({db:"xz",ip:"xxx"})

for of: 简化for循环

如何: for(var p of products){

p会依次取出products中的每个商品

}

class: 简化了OOP

封装:

对象直接量:

var obj={

属性名:变量名, 如果属性名和变量名相同可只写一个

方法名(){

...this.属性名...

}

}

创建一个类型:

class Student {

//一个类型由2部分组成:

//构造函数: 定义属性结构，创建子对象

constructor (sname,sage){

this.sname=sname;

this.sage=sage;

}

//原型对象: 保存子对象共用的API

intr (){

...

}

}

继承:

class Father{  
 ...  
 }

class Child extends Father{

constructor(参数列表){

super(参数列表)

}

}

静态方法:

class xxx{  
 static 方法名(){}  
 }

访问器属性:

class xxx{

constructor(..., age, ...){  
 Object.defineProperties(this,{  
 \_age:{  
 writable:true,

enumerable:false  
 }  
 })

this.age=age;  
 }

//保存在prototype中

get age(){ return this.\_age }

set age(val){

if(val符合要求)

this.\_age=val

else 报错

}

}

使用方法和之前完全一样

Promise: 代替异步调用中的回调函数

异步调用的回调函数的问题: 回调地狱

回调地狱的根本原因: 过早的将下一个函数提前交给上一个函数

为什么使用Promise: 避免回调地狱

何时: 只要多个异步调用必须先后执行时，就用promise

如何: 2步:

1. 定义函数时:

function prev(){

return new Promise(function(open)=>{

//封装原来任务的代码

//决定何时打开开关: open()

})

}

2. 调用时：

prev().then(next)

next一定会等待prev执行完，open()后才能执行

补充: Promise

错误处理:

定义时: return new Promise(function(open,err){  
 //如果正常执行完，调用open()  
 //如果出错，调用err("错误消息")  
 })

调动时:

fun1.then(fun2).then(fun3). ..... .catch(function(errMsg){

输出: errMsg

})

等待多个异步任务完成后才执行:

要求: 每个异步任务都是一个Promise对象

调用时: Promise.all([ task1(), task2(),...]).then(fun)

正课:

1. 什么是DOM

2. DOM Tree

3. 查询

4. 修改

1. 什么是DOM: Document Object Model

什么是: 专门操作网页内容的API

DOM标准: w3c制定的DOM API标准

为什么: 为了统一所有浏览器操作网页内容的API

何时: 只要操作网页内容，都用DOM API

几乎所有浏览器100%兼容DOM标准

2. DOM Tree:

什么是: 网页中所有内容，在内存中，都是保存在一棵树形结构中。

为什么: 树形结构是最直观的表现上下级包含关系的结构

如何: 浏览器读取网页内容时，自动在内存中创建DOM树

根节点: 一个网页只有唯一的一个树根节点: document

所有网页内容都包含在document对象下

节点对象: 网页中每项内容(元素, 属性, 文本)都是一个节点对象

所有节点对象都有的三个属性:

nodeType: 节点类型: 用数字代表节点类型

何时: 只要判断节点的类型时

包括:

document 9

element 1

attribute 2

text 3

问题: 无法进一步区分标签名

nodeName: 节点名称:

何时: 只要进一步判断节点名称时

包括:

document #document

element 全大写的标签名: BUTTON

attribute 属性名

text #text

nodeValue: 节点值:

何时: 几乎不用

包括:

document null

element null

attribute 属性值

text 文本内容

概述: 5件事: 增，删，改，查，事件绑定

1. 查找: 4种:

1. 无需查找可直接获得的节点: 4种

document.documentElement html

document.head head

document.body body

document.forms[id/i] <form>

2. 按节点间关系查找:

何时: 如果已经获得一个节点对象，找周围的节点时

如何:

节点树: 包含所有节点对象(元素,文本...)的树结构

2大类:

1. 父子: 4个

node.parentNode 获得node的父节点

node.childNodes 获得node的所有直接子节点

node.firstChild 获得node的第一个直接子节点

node.lastChild 获得node的最后一个直接子节点

2. 兄弟: 2个

node.previousSibling 前一个兄弟

node.nextSibling 下一个兄弟

节点树的问题: 会受到看不见的空字符的干扰

解决:

元素树: 仅包含元素节点的树结构

2大类:

1. 父子: 4个

node.parentElement 获得node的父元素

node.children 获得node的所有直接子元素

node.firstElementChild 获得node的第一个直接子元素

node.lastElementChild 获得node的最后一个直接子元素

2. 兄弟: 2个

node.previousElementSibling 前一个兄弟元素

node.nextElementSibling 下一个兄弟元素

说明: 元素树其实只是节点树的子集:

强调: childNodes和children

返回: 类数组对象，动态集合

什么是动态集合: 不实际存储属性值，每次访问集合都重新查找DOM树

缺: 反复访问集合，会导致反复查找DOM树

遍历时: for(var i=0,len=children.length;i<len;i++){

优: 首次查找返回速度快，因为只需要返回指定的数据即可，不用返回完整数据。

鄙视: 用递归的方法遍历一个父节点下的所有后代节点

2步:

1. 定义函数，仅遍历直接子节点

2. 对每个直接子节点调用和父元素完全相同的方法

3. 按HTML查找:

什么是: 按元素的HTML特征查找元素

何时: 只要暂时没有获得元素，要进行首次查找时

如何: 4种:

1. 按id查找:

var elem=document.getElementById("id");

强调: 1. 只能用document调用

2. 只返回一个元素

2. 按标签名查找:

var elems=parent.getElementsByTagName("标签")

强调: 1. 可在任意父元素上调用

2. 返回动态集合

3. 不仅查找直接子节点，且在所有后代中查找

3. 按name属性查找:

何时: 专门查找表单中的元素

var elems=document.getElementsByName("name")

强调: 1. 只能用document调用

2. 返回动态集合

4. 按class属性查找:

var elems=parent.getElementsByClassName("class")

强调: 1. 可在任意父元素上调用

2. 返回动态集合

3. 不仅查找直接子节点，且在所有后代中查找

4. 不要求完全匹配，只要元素上有一个class和条件匹配即可

问题: 步骤太繁琐

4. 用选择器查找:

2个:

1. 如果只查找一个元素

var elem=parent.querySelector("选择器");

2. 如果查找多个元素

var elems=parent.querySelectorAll("选择器");

强调: 1. 可在任意父元素上调用

2. 返回非动态集合

什么是非动态集合: 实际存储属性值，即使反复访问集合，也不会导致重复查找DOM树

优: 即使反复访问，也不会导致重复查找DOM树

缺: 首次查找慢

按HTML查找和按选择器查找如何选择:

特点:

1. 按HTML查找返回动态集合

按选择器查找返回非动态集合

2. 按HTML查找效率高

按选择器查找效率低

3. 按HTML查找繁琐

按选择器查找极其简单

如何选择:

1. 如果查找条件简单: 首选按HTML查找

2. 如果查找条件复杂: 首选按选择器查找

4. 修改:

内容:

原始HTML代码片段: elem.innerHTML

文本内容: elem.textContent

1. 去掉内嵌的标签

2. 将转义字符翻译为正文

问题: IE8 不支持elem.textContent

表单元素的内容: elem.value

属性:

样式:

正课:

1. 修改

2. 添加/删除

1. 修改: 3种:

内容: 3种:

.innerHTML

.textContent

.value

属性: 3种:

HTML标准属性: 2种方式:

1. 核心DOM: 兼容所有结构化文档的DOM API

4个API:

elem.getAttribute("属性名")

elem.setAttribute("属性名","属性值")

elem.hasAttribute("属性名")

elem.removeAttribute("属性名")

优: 几乎万能

缺: 繁琐

2. HTML DOM: 专门针对HTML网页的简化版API

已经将所有标准属性定义在了elem元素对象内

可用.直接访问

elem.属性名

elem.属性名=属性值

elem.属性名!==""

elem.属性名=""

特殊: elem.class => elem.className

三大状态属性: disabled checked selected

不能用核心DOM API操作, 因为他们都是bool类型

可用HTML DOM直接用.访问

自定义扩展属性:

什么是: HTML标准中未定义的，开发人员根据业务需要，自行添加的属性

何时: 只要访问元素时，同时需要获得该元素对应的业务数据时，都会提前将业务数据保存在当前元素上

如何:

1. 核心DOM访问:

2. 用HTML5新方式:

定义时，HTML中: <ANY data-属性名="值"

访问时: elem.dataset.属性名

dataset可自动收集所有data-开头的自定义扩展属性

样式: 2种:

1. 内联样式:

修改: elem.style.css属性="值"

强调: css属性名去横线变驼峰

elem.style.listStyleType

.fontSize

.backgroundColor

获取样式: 不能用elem.style，elem.style只能获得内联样式。

如何获得一个元素的完整样式:

获得计算后的样式: 最终应用到一个元素上的完整样式

何时: 只要获得任意元素的样式时，都要用计算后的样式

如何: var style=getComputedStyle(elem)

style.css属性

强调: 计算后的样式，都是只读，不能修改

因为计算后的样式，来源不确定，影响范围不确定

2. 修改样式表中的共享样式:

获得样式表对象: 网页中的每个样式表都是一个对象

var sheet=document.styleSheets[i];

获得样式表对象中的规则: cssRule

var rule=sheet.cssRules[i]

如果是修改keyframes中的css属性，则还要继续找下级cssRule

var subRule=rule.cssRules[i]

修改规则中的css属性:

rule.style.css属性=值

问题: style.xxx一句话只能修改一个css属性，如果需要同时修改多个css属性，代码繁琐

解决: 将多个css属性，提前定义为class，用程序操作class。

2. 添加/删除:

添加: 3步:

1. 创建空元素:

var elem=document.createElement("标签名");

比如: var a=document.createElement("a")

<a></a>

2. 添加必要的属性

a.href=url

a.innerHTML=文本

<a href="url">文本</a>

3. 将新元素添加到指定父元素下

parent.appendChild(elem) 将新元素elem追加到parent的所有子元素之后

parent.insertBefore(elem,child) 将新元素elem插入到parent下现有child元素之前

parent.replaceChild(elem,child) 用新元素elem代替parent下现有child元素

正课:

1. 添加和删除:

2. HTML DOM常用对象

3. 打开和关闭窗口

4. history

5. location

6. 定时器

1. 添加和删除:

优化: 尽量减少操作DOM树的次数

为什么:

HTML页面加载原理:

html -> DOM Tree

↓

Render Tree→ layout →paint

↑

css -> COM

后来，只要修改DOM树，就会需要重新layout和paint

如何: 2种:

1. 如果同时添加父元素和子元素:

那么，应该在内存中，先将所有子元素添加到父元素，最后，再一次性将父元素添加到DOM Tree

2. 如果父元素已经在页面上，需要添加多个平级子元素

应该用文档片段

什么是: 内存中，临时保存多个平级子元素的虚拟父元素对象

何时: 添加多个平级子元素

如何: 3步:

1. 创建文档片段:

var frag=document.createDocumentFragment();

2. 将子元素添加到文档片段中

frag.appendChild(child)

3. 将文档片段一次性添加到父元素上

parent.appendChild(frag)

强调: frag将子元素添加到页面后，自动释放

删除: parent.removeChild(child)

child.parentNode.removeChild(child)

2. HTML DOM常用对象:

Image 创建: var img=new Image();

Select/Option

Select:

属性:

.selectedIndex获得select中当前选中的option的位置

.options 获得select下所有option的集合

.options.length 获得select下所有option的个数

.length => .options.length

清空所有option: .length=0;

方法:

.add(option) 代替appendChild

问题: 不支持fragment

.remove(i)

Option:

创建: var opt=new Option(text,value);

属性: ... ...

Table/...

管着行分组:

添加: var thead=table.createTHead();

tbody=table.createTBody();

tfoot=table.createTFoot();

删除: table.deleteTHead();

table.deleteTFoot();

获取: table.thead

table.tfoot

table.tbodies[i]

行分组管着行:

创建: var tr=行分组.insertRow(i);

固定用法: 在末尾追加新行: .insertRow()

在开头插入: .insertRow(0)

删除: 2种:

1. 行分组.deleteRow(i)

强调: 因为用行分组删除，所以i要求是行分组内的下标位置。

问题: 行在行分组内的位置无法自动获得

只能获得行在整个表中的位置: tr.rowIndex

解决:

2. table.deleteRow(tr.rowIndex) ——推荐的

获得: 行分组.rows[i]

行管着格:

创建: var td=tr.insertCell(i);

固定用法: 末尾追加新格: tr.insertCell()

强调: 只能创建td，不能创建th

删除: tr.deleteRow(i)

获取: tr.cells[i]

Form/element

获取: var form=document.forms[i/id/name];

属性:

.elements 获得表单中的所有表单元素

.elements.length 获得表单中表单元素的个数

.length => .elements.length

方法:

.submit() 手动提交

.reset() 重置

表单元素:

获取: var 表单元素=form.elements[i/id/name]

更简化: 如果表单元素有name属性

可: form.name

方法: elem.focus() 让elem获得焦点

elem.blur()

BOM: Browser Object Model

什么是: 专门操作浏览器软件/创建的API

问题: 没有标准，兼容性差

包括:

window: 2个角色

1. 包含所有ES, DOM和BOM的API

2. 代替ES中的global充当全局作用域对象

history

location

navigator

document

event

screen

3. 打开和关闭窗口:

打开窗口(新链接): 4种

1. 在当前窗口打开, 可以后退

html: <a href="url" target="\_self">

js: /\*window.\*/open("url","\_self")

2. 在当前窗口打开, 禁止后退

js: location.replace("新url")

3. 在新窗口打开，可打开多个

html: <a href="url" target="\_blank">

js: open("url","\_blank")

4. 在新窗口打开，只能打开一个

html: <a href="url" target="自定义窗口名">

js: open("url","自定义窗口名")

原理: 每个窗口都有一个隐藏的name属性

在打开窗口时，通过target指定的

浏览器规定: 相同name属性的窗口只能打开一个

后打开的会替换先打开的

如果:

\_self: 自动获得当前窗口的name，用于新窗口

结果: 新窗口，替换当前窗口

\_blank: 不指定窗口名，浏览器会随机分配窗口名

关闭: /\*window.\*/close();

4. history:

什么是: 每个窗口中保存当前窗口打开后，成功访问过的url的历史记录栈

只开放了一个API: history.go(n) 模拟前进/后退

前进一步: history.go(1)

刷新: history.go(0)

后退一步: history.go(-1)

5. location:

什么是: 专门保存当前窗口正在打开的url的对象

如何:

属性:

.href 获取或设置当前窗口的url

.protocol: 协议

.host: 主机名:端口号

.hostname: 主机名

.port: 端口

.pathname: 相对路径

.hash: #锚点地址

.search: ?查询字符串参数

鄙视: 解析查询字符串为对象

方法:

1. 在当前窗口打开新链接，可后退:

location.assign("新url")

=> location.href="新url"

=> location="新url"

2. 在当前窗口打开新链接，禁止后退:

location.replace("新url")

3. 刷新: 2种:

1. 普通刷新: 优先从浏览器本地缓存中获取文件，本地没有，才去服务端重新下载

history.go(0)

location.reload(false)

F5

2. 强制刷新: 每次忽略缓存，直接从服务器下载最新资源

location.reload(true)

6. 定时器: 2种:

1. 周期性定时器:

什么是: 让程序每隔一段时间间隔，反复执行一项任务

如何: 3件事:

1. 定义任务函数:

2. 启动定时器: timer=setInterval(task, 间隔ms)

每启动一个定时器，都会为定时器添加一个唯一的序号，从1开始，递增，连续不重复

只有在启动定时器时，才能获得序号

强调: timer经常跨函数使用，所以一般为全局变量

3. 停止定时器: clearInterval(timer)

2种:

1. 手动点击按钮停止

2. 自动停止:

在任务函数task中，判断临界值。

如果临界值达到，则自动停止定时器

2. 一次性定时器:

正课:

1. 定时器:

2. navigator:

3. \*\*\*event:

1. 定时器:

一次性定时器: 让程序先等待一段时间，再延迟执行一项任务。执行后，自动停止，不再反复执行。

如何: 3件事:

1. task()

2. 启动定时器: var timer=setTimeout(task, 延迟ms);

3. 停止定时器: clearTimeout(timer);

定时器的原理:

启动定时器时，创建定时器对象，保存要执行的函数

定时器对象会按指定时间，自动将要执行的函数，加入回调队列

回调队列必须等待主程序都执行完，才能开始执行。

总结: 定时器中的回调函数，一定会等待主程序执行完，才能开始执行

鄙视:

1. var a;

function task(){

a=b=10;

}

setTimeout(task,0);

console.log(a);//?

console.log(b);//?

//主程序执行完之后: task()

2. for(var i=0;i<3;i++){

setTimeout("console.log(i)",0)

}//i=3

2. navigator:

什么是: 保存浏览器配置信息的对象

何时: 只要获得浏览器配置信息

包括:

1. cookieEnabled: 判断当前浏览器是否启用cookie

cookie:

什么是: 客户端持久存储用户私密信息的小文件

为什么: 程序中的数据都是保存在内存中的，都是临时的。一旦程序关闭，内存释放，数据丢失。

何时: 只要希望在客户端持久存储数据

2. plugins: 包含当前浏览器安装的所有插件信息的集合

插件: 为浏览器添加新功能的小软件

如何判断: navigator.plugins["完整插件名"]!==undefined

3. userAgent:

什么是: 包含浏览器名称和版本号的字符串

何时使用: 只要判断浏览器的名称和版本号时

如何:

3. \*\*\*event:

什么是事件: 浏览器自动触发的或用户手动触发的页面内容状态的改变

事件处理函数: 当事件发生时，自动执行的函数

何时: 只要希望事件发生时，执行一项任务

如何: 3种:

1. 在HTML中:

html: <ANY on事件名="调用js中的函数/js语句"

js: function 函数(){  
 ....

}

2. 在js中绑定，一个事件只能绑定一个处理函数

elem.on事件名=function(){

...

}

问题: 一个事件只能绑定一个处理函数

因为是赋值方式绑定

3. 在js中绑定，一个事件可绑定多个处理函数，也可随时移除

elem.addEventListener("事件名",function(){...})

elem.removeEventListener("事件名",原处理函数名)

强调: 移除时，必须找到原函数对象

如果一个处理函数可能被移除，则绑定时，不能使用匿名函数，必须使用有名称的函数

事件模型: 从事件发生到所有处理函数执行完，经过的过程

DOM标准: 3个阶段:

1. 捕获: 由外向内，记录各级父元素上绑定的处理函数

2. 目标出发: 先触发目标元素上的事件

目标元素: 最初触发事件的第一个元素

3. 冒泡: 由内向外，按照捕获阶段的反向顺序，依次触发父元素上绑定的处理函数

所有浏览器的开发者都认为，点在内层元素上，相当于点在外层元素上

事件对象:

什么是: 专门保存事件信息，并提供操作事件的API

何时: 只要获得事件相关的信息，或要操作事件时

如何:

创建: 事件发生时，自动创建

获得: DOM标准: 事件对象总是作为处理函数的第一个参数，默认传入

API:

1. 取消冒泡/停止蔓延:

e.stopPropagation()

2. 利用冒泡:

优化: 尽量减少事件监听的个数

为什么: 浏览器中所有事件监听对象都集中存储在一个大的集合中。触发事件时，使用遍历的方式，查找要执行的处理函数。所以，事件监听对象的个数，直接影响了触发事件的效率

如何: 只要多个子元素，绑定相同的事件时，都要只在父元素上绑定一次，所有子元素都能触发

2个难题:

1. 获得实际触发事件的目标元素:

错误: this->父元素

正确: e.target

2. 鉴别目标元素是不是想要的

3. 取消事件/阻止默认行为:

e.preventDefault();

3种:

1. 用a元素当按钮用时:

2. 自定义表单提交时: 2种:

input button btn.onclick form.submit()

input submit form.onsubmit e.preventDefault()

3. HTML5 拖拽API: 都要先阻止默认行为，再定义自己的操作。

4. 鼠标位置:

相对于屏幕左上角的位置: e.screenX e.screenY

相对于文档显示区左上角的位置: e.clientX e.clientY

相对于当前元素左上角的位置: e.offsetX e.offsetY

正课:

1. 什么是jQuery:

2. 如何使用jQuery:

3. 查找:

1. 什么是jQuery:

第三方的执行DOM操作的极简化的函数库

第三方: 下载

执行DOM操作: 学习jQuery还是在学DOM:

1. 增删改查

2. 事件绑定

3. 动画

4. ajax

极简化: jQuery是DOM操作的终极简化版本

函数库: jQuery中一切都是函数

为什么: 2大原因:

1. 简化:

2. 解决了绝大多数兼容性问题: 凡是jQuery让用的，都没有兼容性问题:

鄙视: 如何快速学习一项新技术/框架:

3步:

1. 上官网，看是什么:

2. 上官网，看快速入门:

3. 上官网，下载手册:

2. 如何使用:

下载:

版本:

1.x: 兼容旧浏览器:

未压缩版: 保持完备注释，代码格式，见名知义的变量名的版本。

特点: 优: 可读性好，便于学习和研究

缺: 体积大，不便于下载

何时: 学习和开发环境中

压缩版: 3件事: 去注释， 去掉代码格式，将变量名极简化

特点: 优: 体积小，下载快

缺: 毫无可读性

何时: 生产环境

2.x: 不再兼容旧浏览器

3.x: 不再兼容旧浏览器，加入了很多新特性:

鄙视: jQuery 3.x新特性:

1. jQuery自身版本，都运行在严格模式下

2. 支持for...of循环，代替$.each()

3. 新动画API: requestAnimationFrame()

4. 支持promise:

2种引入方式:

1. 将jquery.js文件下载到项目本地文件夹保存

2. 使用CDN网络: 根据用户到网络中任意一台服务器的网络状况，自动选择最优的服务器下载资源。

原理:

引入jquery.js，其实是向window中引入一种新的类型

1. 构造函数: 创建jQuery类型的子对象

何时: 只要使用jQuery都要先创建jQuery类型的子对象

为什么: 因为普通DOM元素对象，不能使用jQuery的简化版API。

如何创建: 2种:

1. 通过查找创建:

var jq=$("selector")

查找到满足selector条件的所有元素，保存到新创建的jQuery对象中

2. 将现有元素包装进jQuery对象中

var $elem=$(DOM元素)

什么是jQuery对象: 封装查找到的DOM元素，并提供操作DOM元素的API 的类数组对象

2. 原型对象: 保存所有jQuery类型子对象都能使用的API

jQuery API三大特点:

1. 一个API两用: 如果提供了新值，就修改；没提供新值，就获取。

2. 自带遍历: 对jQuery对象调用一次api，会自动应用到jQUery对象内每个DOM元素上

3. 多数API都返回正在操作的jQuery对象

如果对同一个jQuery对象执行多个操作，可用链式操作。

3. 查找: 2种:

1. 按选择器:

jQuery支持所有CSS的选择器。并扩展了少量新选择器。

基本选择器: 5个: 同CSS

id elem .class \* 选择器1,选择器2,...

层级选择器: 4个: 同CSS

后代 子代 + ~

过滤选择器:

子元素过滤: 4个: 同CSS

:first-child :last-child :nth-child(i) :only-child

基本过滤: jQuery中新增

:first/last :eq/lt/gt(i) :even/odd

属性过滤: 同CSS

[属性名]

[属性名=值]

[属性名^=值]

[属性名$=值]

[属性名\*=值]

[属性名!=值]

selector1selector2selector3....

可见性过滤: 2个

:hidden 专门选择/判断隐藏的元素

只能选择2种: type="hidden" 和 display:none

:visible 专门选择/判断显示的元素

内容过滤: 4个: jQuery中添加

第一对儿: 按有没有子内容，选择/判断元素

:parent :empty

第二对儿:

1. 按内容文本选择/判断元素

:contains(文本)

2. 按子元素特征选择/判断元素

:has(selector)

状态过滤: 4个: 同css

:enabled :disabled :checked :selected

表单元素过滤: jQuery中添加

专门用于在表单中，选择表单元素:

:[type] : 每种type都对应着一个专门的选择器

:text :password :submit :reset :button ...

:input : 选择所有表单元素:

input textarea button select

2. 按节点间关系:

正课:

1. 修改

2. 按节点间关系查找

3. 添加，删除，替换，克隆

4. 事件绑定

1. 修改:

内容:

HTML代码片段: .html() .innerHTML

纯文本内容: .text() .textContent

表单元素的值: .val() .value

属性:

HTML标准属性: .attr("属性名"[,"值"]) .属性名

相当于: .getAttribute/setAttribute

同时修改多个属性:

.attr({

属性名:值,

属性名:值,

... : ...

})

状态属性: .prop("属性名"[,bool]) .属性名

自定义扩展属性: 2种:

可读取任意自定义扩展属性

.attr("自定义属性名")

只能读取data-\*开头的自定义扩展属性

.data("自定义属性名") .dataset.属性名

样式:

获取/修改css属性:

.css("css属性名","值")

如果修改css属性时，.css相当于.style.css属性名

如果获取css属性值时，.css自动调用getComputedStyle

设置class属性，批量应用样式

.addClass("class")

.removeClass("class")

.hasClass("class")

.toggleClass("class")

等效于: if(hasClass) removeClass else addClass

2. 按节点间关系查找: 2大类关系:

1. 父子:

.parent() .parentNode

.children(["selector"]) .children 只查找直接子元素

.find("selector") 在所有后代中查找

.children(":first") .firstElementChild

.children().first()  
 .children(":last") .lastElementChild

.children().last()

2. 兄弟:

.prev() .previousElementSibling

.prevAll(["selector"]) 之前所有兄弟

.next() .nextElementSibling

.nextAll(["selector"]) 之后所有兄弟

.siblings(["selector"]) 除我之外的前后所有兄弟

3. 添加，删除，替换，克隆

添加:

1. 用$创建新元素: var $elem=$("html代码片段")

2. 将$elem添加到DOM树

4种:

1. 末尾追加: $parent.append($elem) .appendChild

$elem.appendTo("parent selector") //return $elem

2. 开头插入: $parent.prepend($elem)

$elem.prependTo("parent selector") //return $elem

3. 插入到指定子元素前: $child.before($elem)

4. 插入到指定子元素后: $child.after($elem)

其实都可简化为一步:

$parent.append("html片段")

$("html片段").appendTo($parent)

删除: $(...).remove();

替换: 2个:

$(现有元素).replaceWith(新元素)

$(新元素).replaceAll(现有元素)

克隆:

$(...).clone() 浅克隆: 仅复制属性和样式，不复制行为

$(...).clone(true) 深克隆: 即复制属性和样式，又复制行为

4. 事件绑定:

DOM: .addEventListener("事件名",handler)

.removeEventListener("事件名",handler)

鄙视: jQuery中共有几种事件绑定方式，分别有什么特点

1. .bind/unbind("事件名",handler)

单纯简化addEventListener/removeEventListener

unbind() 三个重载:

unbind("事件名",handler) 移除一个事件上绑定的一个指定处理函数

unbind("事件名") 移除一个事件上绑定的所有处理函数

unbind() 移除所有事件上绑定的所有处理函数

2. .one() 特殊的bind，绑定事件后，只触发一次，执行后，自动解绑。

3. .delegate() 事件委托, 其实就是简化版利用冒泡

何时: 只要利用冒泡，就用delegate

如何: $parent.delegate("selector","事件名",handler)

其中: 2个简化:

1. "selector" 用于筛选符合条件的子元素才能触发事件。——自己不用再写if判断

2. 又让this重新指向e.target

鄙视: bind vs delegate

1. 绑定位置:

.bind() 直接绑在目标子元素上

.delegate() 只在父元素上绑定一次

2. 事件监听的个数:

.bind() 导致更多的监听

.delegate() 只有一个监听

3. 对动态添加的新元素:

.bind() 必须重复绑定，才能让新元素获得事件

.delegate() 无需反复绑定，新元素也可自动获得父元素上的事件

.undelegate() 解绑父元素上共享的处理函数

4. .live/die 废弃:

5. .on/off 统一了bind和delegate

重载:

.on("事件名",handler) 代替bind

.on("事件名","selector",handler) 代替delegate

6. .事件名(handler) 其实只是.bind的简化

问题: 1. 不能代替delegate简化利用冒泡

2. 不是所有事件都有简化。

正课:

1. 事件:

2. 动画:

3. 类数组对象操作:

4. 添加自定义API:

1. 事件:

页面加载后执行: 2种:

1. 仅DOM内容(html,js)加载后就提前执行:

DOMContentLoaded

何时: 如果一些初始化操作，不必等待css和图片，就可执行时，应放在DOMContentLoaded中提前执行:

比如: 事件绑定!

如何: $(document).ready(function(){ ... })

简化: $().ready(function(){ ... })

更简化: $(function(){ ... })

2. 在所有页面内容(html,css,js,图片)加载后执行:

window.onload=function(){}

何时: 如果一个操作，必须等待css和图片加载后才能执行，应放在onload中

如何: $(window).load(function(){ ... })

鼠标事件:

mouseover mouseout

问题: 反复进出子元素，会反复触发父元素上的事件

解决: mouseenter mouseleave

即使反复进出子元素，也不会反复触发父元素上的事件

如果同时绑定enter和leave，可简写为hover

$().hover(

function(){ ... }, //mouseenter

function(){ ... } //mouseleave

)

如果两个处理函数可统一为一个，则可省略另一个

模拟触发: 手动调用一个元素的事件处理函数

如何: $(...).trigger("事件名")

简化: 事件绑定函数也可两用:

$(...).事件名([function(){ ... }])

2. 动画:

1. 简单动画: 3组固定效果:

1. 显示隐藏: .show() .hide() .toggle()

强调: 不带参数, 用display瞬间显示隐藏

可专门用于代替.style.display=xxx

如果希望动画效果: 应加入时间参数

问题: 1. 用js程序的定时器实现的，效率不如css

2. 动效是固定的，几乎不可维护

解决: class + transition

2. 上滑下滑: .slideUp() .slideDown() .slideToggle()

3. 淡入淡出: .fadeIn() .fadeOut() .fadeToggle()

2. 万能动画:

什么是: 可对任意数值类型的css属性添加动画效果

何时: 只要用程序控制修改css属性值，实现动画效果

如何:

$(...).animate({

css属性名:目标值,

... : ... ,

},持续时间)

强调: 只能支持单个数值的css属性

停止动画: $(...).stop()

问题: $(...).stop()只能停止当前正在播放的一个动画

队列中后续动画依然继续播放

解决: $(...).stop(true) 停止动画，并清空队列

判断/选取正在播放动画的元素: :animated

动画播放后执行:

所有动画API的最后一个参数都是回调函数，在动画结束后自动执行。

排队和并发:

1. 并发: 多个css属性同时变化

如何: 只要放在一个animate内的多个css属性，都是并发变化

2. 排队: 多个css属性先后依次变化

如何: 对一个元素，反复调用多次动画API，多次动画API以排队方式执行。

其实: 调用动画API，仅是将动画加入执行队列的意思，不是立刻执行。

延迟执行: $(...).delay(ms).动画API

3. 类数组对象操作:

jQuery对类数组对象提供了简化版API:

静态方法: 通过$直接调用，即支持数组，又支持类数组对象

$.each(类数组对象/数组, function(i, elem){

/\*对每个元素值执行相同的操作\*/

})

var i=$.inArray(要查找的元素, 数组/类数组对象[,fromi])

jQuery类型的实例方法:

$(...).each()

$(...).index()

简化: 如果在同一个父元素内检查位置

$(子元素).index();

鄙视: $.each() 和 $(...).each

1. 存储位置:

$.each() 是静态方法，直接定义在jQuery构造函数上

$(...).each() 是实例方法，保存在jQuery类型的原型对象中

2. 支持的对象类型:

$.each() 可遍历任意数组和类数组对象

$(...).each() 只能被jQuery类型的子对象调用

4. 添加自定义API:

如果jQuery的简化版API不够用，则可手动添加自定义API

如何: jQuery.fn.自定义方法=function(){

//this->

}

正课:

1. jQuery UI

2. 自定义插件

3. 第三方

插件/组件: 拥有独立的界面和行为的可重用的小功能

为什么: 很多项目之间，大部分功能，都是很类似的。

何时: 只要在项目中发现反复使用的小功能，都要将小功能封装为插件/组件

包括: 3种来源：

1. jQuery官方: jQuery UI

2. 第三方:

3. 自定义封装:

1. jQuery官方: jQuery UI

下载并引入:

文件夹结构:

css/

jquery-ui.css

images/

js/

jquery.js

jquery-ui.js

引入:

link rel="stylesheet" href="css/jquery-ui.css"

script src="jquery.js"

script src="jquery-ui.js"

如何使用:

1. 效果:

jQuery UI重写了jQuery中一些API，添加了额外的效果

1. 重写了.animate方法，使其支持颜色动画

2. 为addClass添加了动画效果

3. 为简单动画，添加了更多特效

2. 交互:

3. 小部件/组件:

如何:

1. 引入插件的css文件

2. 按照插件的约定，定义HTML内容

3. 引入jquery和jquery UI的js

4. 在自定义脚本中，找到要应用插件的父元素，调用插件API

侵入性: 插件根据自己的需要，自动的向元素中添加辅助的样式属性和行为

2. 自定义插件:

前提: 已经用普通的HTML，CSS，JS实现了

如何: 2种:

1. jQuery UI的侵入方式:

1. 将CSS提取到单独的css文件中保存

强调: css中的选择器避免使用id, 元素。优先使用class和自定义扩展属性

2. 定义插件.js文件:

判断是否提前加载了jquery.js  
 向jQuery.fn中添加插件API:

向当前元素下的子元素侵入class和自定义扩展属性

为当前元素下的子元素绑定事件

3. 在页面中:

按插件的要求，定义html内容格式

引入插件css  
 引入jquery.js，插件.js

自定义脚本中，找到插件父元素，调用插件API

2. bootstrap DIY方式:

1. 同jQuery UI方式

2. 定义插件.js文件:

判断是否提前加载了jquery.js

在整个页面中查找符合条件的元素，自动绑定事件

3. 在页面中:

引入插件css

按插件的要求，定义html内容格式

手动添加class和自定义扩展属性  
 引入jquery.js，插件.js

在引入插件js时，插件js会自动查找当前网页中符合条件的元素，添加行为

正课:

1. 第三方插件

2. jQuery Ajax API:

3. \*\*\*\*\*跨域:

1. 第三方插件:

1. 验证:

2. 文件上传:

3. wysiwyg:

4. masonry:

2. jQuery中的ajax API

$.ajax({

url:

type:

data:

dataType:

beforeSend:function(xhr请求对象){

}

success:function(data){ //当成功返回结果时

}

error:function(err){ //请求出错时

}

complete:function(){ //请求完成

}

}).then(data=>{

//data是服务器端返回的数据(json对象或字符串)

})

$.get("url",data,success,"dataType")

$.getJSON("url",data,success)

$.getScript()

$(父元素).load("url")

$.post("url",data,success,"dataType")

3. \*\*\*\*\*跨域:

什么是: 发起请求的域名地址和要请求的目标地址不在同一域名下

何时:

1. 自己的应用程序，要请求别人网站上的数据

比如: 股票，天气，。。。

2. 自己的项目太大，前端和后端的应用程序，也分数不同域名之下

3. 自己公司的多个系统之间，互相请求数据

包括:

1. 端口号不同

2. 协议不同

3. 二级域名不同

4. 域名不同

5. 即使同一主机的IP和域名之间互访

问题: xhr ajax请求不允许跨域！

解决:

除了xhr外，有很多元素/属性可以跨域:

比如: img.src link.href script.src

1. jsonp(json with padding)方式跨域: (通过script.src)

方式一:

客户端: 不要使用xhr，用script标签请求php

服务端: 不只返回数据

而是将数据填充在一段js代码中，返回

问题: 服务器端返回的语句是固定的，客户端无法灵活改变。

解决:

方式二:

客户端: 定义一个接受参数的函数

用script标签请求php

服务端: 只返回一条函数调用语句

函数名与客户端函数名保持一致

问题: 函数名必须和服务端统一

解决:

方式三:

客户端: 定义一个接受参数的函数

用script标签请求php, 携带一个参数callback，值为函数名

服务端: 从请求中获得callback参数值(函数名)

只返回一条函数调用语句

将函数名和数据拼接为一条调用的js语句

问题: 只能在页面加载时就执行，无法自行选择触发的时机

解决:

方式四: 客户端动态创建script添加到页面，用完，自动删除script

$.ajax是支持jsonp的

2. 服务端跨域:

修改响应头，添加允许跨域的地址:

header("Access-Control-Allow-Origin: 请求来自的地址");