Day01

张东

zhangdong@tedu.cn

学习方法:

1. 拿到上届笔记,前一天晚上预习，第二天拿着笔记上课

思维导图: 快速查找API

2. 案例:

注释->每行注释一分钟->公布->不等

课下: 3遍:

1. 注释和代码 抄

2. 留注释，写代码

3. 自己可以把注释写回来

3. 学的好坏:3档:

1. 老师边写注释，学生边写代码

2. 老师写完注释，在规定时间内，可以码出代码

3. 看着老师的注释，写不出代码

4课下关注

1.书: 犀牛书

2.微信: 前端大全

3.js1k: 毕业时的个人案例-游戏，算法

正课:

1.\*\*\*数组:

1.什么是: 内存中连续存储多个数据的数据结构，再起一个统一的名字

2.为什么: 普通的变量只能存储一个数据

程序=数据结构+算法:

算法: 解决问题的步骤

数据结构: 数据在内存中的存储结构

好的数据结构可以极大的提高程序的执行效率

3.何时: 只要存储多个连续的数据

4.如何使用:

1.创建: 3种:

1. 创建空数组: 2种方法:

1. 数组直接量: var arr=[];

2. 用new: var arr=new Array();

新建 数组

2.何时: 在创建数组时，还不知道数组中的元素内容时

2. 创建数组同时，初始化输出中的元素:

1. 数组直接量: var arr=[元素1,元素2,...];

2. 用new: var arr=new Array(元素1,元素2,...);

何时: 在创建数组时，已经知道数组的元素内容

3. 创建n个空元素的数组:

用new: var arr=new Array(n);

2.数组是引用类型的对象: vs 原始类型的值对比

原始类型的值: 数据直接保存在变量本地的数据类型

number, string, boolean, null, undefined

引用类型的对象: 数据无法直接保存在变量本地的数据类型

如何存储: 在变量之外开辟独立的存储空间

变量中仅保存外部存储空间的地址

回顾: 按值传递:

两个变量间赋值时，或将变量作为参数传入函数时，其实只是将原变量中的值复制一个副本给对方！

结果: 对原始类型的值: 修改新变量，不影响原变量

为什么: 复制的是原始的值

但是，对引用类型的对象: 通过新变量修改对象，等效于直接修改原对象。

为什么: 仅复制地址值，原对象不增加

3.垃圾回收: 引擎会自动释放不再使用的对象的空间

垃圾回收器: 自动记录对象的引用次数，并释放不再被任何变量引用的对象的 小程序

伴随js主程序并行执行。

原理: 创建对象时，每个对象都有一个属性:引用计数器，用来记录引用对象的变量个数

\*如果变量不再使用对象时，要将变量赋值为null\*

对象的引用计数器-1

垃圾回收器会自动寻找引用计数器为0的对象，释放

4.访问数组中的元素:

元素: 数组中每个数据都是一个元素

如何访问: 下标: 数组中唯一标识每个元素存储位置的序号

特点: 从0开始,连续不重复

何时: 只要访问数组元素，只能用下标

如何使用:

语法：数组名[i]

注意：用法和单个变量完全一样！

数组的length属性: 记录了数组中理论上的元素个数

length属性的值永远是最大下标+1

固定套路:

1.获得数组最后一个元素: arr[arr.length-1]

2.获得倒数第n个元素的位置: arr[arr.length-n]

3.在数组末尾追加一个新元素: arr[arr.length]=新值

4.数组缩容: 减小arr.length的数值，会删除结尾的多余元素。

5.遍历数组: 依次访问数组中每个元素，对每个元素执行相同的操作

for(var i=0;i<arr.length;i++){

arr[i]//当前正在遍历的元素

}

特殊注意: 三个不限制:

1. 不限制数组的元素个数:长度可变

2. 不限制下标越界:

获取元素值: 不报错！返回undefined

修改元素指: 1.不报错! 自动在指定位置创建新元素，并且自动修改length

属性为最大下标+1

2.如果下标不连续的数组——稀疏数组

3. 不限制元素的数据类型:

二、 \*\*\*关联数组:（关联数组，也叫hash数组）

什么是索引数组: 下标为数字的数组

什么是关联数组: 可自定义下标名称的数组

为什么: 索引数组中的数字下标没有明确的意义

何时使用: 只要希望每个元素都有专门的名称时

如何使用: 2步:

1. 创建空数组

2. 向空数组中添加新元素，并自定义下标名称

区别：

索引数组 vs 关联数组:

1. 已字符串输出 不能用字符串输出

2. 下标是数字 下标是自定义的字符串

3. length属性有效 length属性失效(=0)

4. 访问元素,都用数组名["下标"]

5. 可用for循环遍历 不能用for循环遍历——for in

遍历关联数组: for in循环

for(var key in hash){

key//只是元素的下标名

hash[key]//当前元素值

}

注意：\*\*\*其实: js底层，一切对象都是关联数组:

比如: 索引数组，window等

三、hash: 散列算法，

什么是：接受一个字符串，并计算出尽量不重复的序号

注意： 不同的字符串，计量计算出不同的序号

相同的字符串，一定计算出相同的序号

原理:

存入元素: 将字符串类型的元素名，交给hash算法，计算出一个尽量不重复的数字。再将元素值，存入数字表示的位置

获取元素: 将要查找的元素名, 交给hash算法，计算出和存入时完全一致的序号，直接到序号标识的位置获取元素值。

区别：

索引 hash数组

遍历 不用遍历

受存储位置影响 和存储位置无关

受数组元素个数影响 和数组中元素个数无关

总结: 今后只要希望快速查找元素时，就用hash数组

Ex:

//定义函数getMax,参数:arr

function getMax(arr){

//从1开始遍历arr中每个元素,同时声明变量max,

初始化为数组第一个元素的值

//如果当前元素>max，就将max改为当前元素值

for(var i=1,max=arr[0];

i<arr.length;

arr[i]>max&&(max=arr[i]),i++);

//(遍历结束)

return max;//返回max

}

测试：var arr=[3,1,5,2,4,7,6];

console.log(getMax(arr));//7\*/

//鄙视题: 去掉数组中重复的元素:

//定义函数unique1,参数:arr

function unique1(arr){

//i从1开始遍历arr中每个元素,同时声明数组r，

初始化一个元素为arr中第0个元素

for(var i=1,r=[arr[0]];i<arr.length;i++){

//j从0开始遍历r数组中每个元素

for(var j=0;j<r.length;j++){

//如果r数组中当前元素等于arr中当前元素，就退出循环

if(r[j]==arr[i]){break;}

}//(遍历结束)

//如果j等于r的length时

//就将arr中当前元素追加到r的结尾

j==r.length&&(r[r.length]=arr[i]);

}//(遍历结束)

return r;//返回r

}

//hash的方法

function unique2(arr){

//声明空数组

var hash=[];

//将arr中第0个元素放入hash作为第一个key

hash[arr[0]]=1;

//i从1开始遍历arr中每个元素,同时声明数组hash，初始化一个元素为arr中第0个元素

for(var i=1;i<arr.length;i++){

//如果hash中不包含arr中当前元素作为key的元素

if(hash[arr[i]]===undefined){

//在hash中添加新元素，key为arr中当前元素,值为1

//为数组中不存在的位置赋值，结果会自动添加新元素

hash[arr[i]]=1;

}

}

//技巧:获取hash中的所有key

var keys=[], i=0;

for(keys[i++] in hash);

return keys;//返回keys

}

/\*测试

var arr=[1,3,1,2,1,2];

//console.log(String(unique1(arr)));

console.log(unique2(arr));//[1,2,3]\*/

/\*

for(var i=0,nums=[];i<100000;i++){

nums[i]=parseInt(Math.random()\*1000);

}

console.time("unique1");

unique1(nums);

console.timeEnd("unique1");

console.time("unique2");

unique2(nums);

console.timeEnd("unique2");

\*/

//定义函数indexOf,参数:arr,val,fromi

function indexOf(arr,val,fromi){

//如果fromi是undefined,就fromi赋值为0

fromi===undefined&&(fromi=0);

//i从fromi开始遍历arr中每个元素

for(var i=fromi;i<arr.length;i++){

//如果当前元素等于val

if(arr[i]==val){

return i;//返回当前元素的位置

}

}//(遍历结束)

return -1;//返回-1

}

//测试:

var arr=[1,2,3,2,1];

//0 1 2 3 4

//从0位置开始找第一个2的位置

console.log(indexOf(arr,2/\*,0\*/));//1

//从2位置开始找下一个2的位置

console.log(indexOf(arr,2,2));//3

//从4位置开始找下一个2的位置，返回-1

console.log(indexOf(arr,2,4));//-1

//从0位置开始找下一个4的位置，返回-1

console.log(indexOf(arr,4/\*,0\*/));

</script>

数组API:

1.数组转字符串:

1. String(arr): 将arr中每个元素转为字符串，用逗号分隔

固定套路: 对数组拍照: 用于鉴别是否数组被修改过

2. arr.join("连接符"): 将arr中每个元素转为字符串，用自定义的连接符分隔

固定套路:

1.将字符组成单词: chars.join("")->无缝拼接

扩展: 判断数组是空数组: arr.join("")==""

2.将单词组成句子: words.join(" ")

3.将数组转化为页面元素的内容:

"<开始标签>"+

arr.join("</结束标签><开始标签>")

+"</结束标签>"

正课:

数组API:

1. 拼接和选取: 不直接修改原数组，而返回新数组！

拼接:

var newArr=arr1.concat(值1,值2,arr2,值3,...)

将值1,值2和arr2中每个元素,以及值3都拼接到arr1的元素之后，返回新数组

其中: arr2的元素会被先\*打散\*，再拼接

选取:

var subArr=arr.slice(starti,endi+1)

选取arr中starti位置开始，到endi结束的所有元素组成新数组返回——原数组保持不变

强调: 凡是两个参数都是下标的函数，都有一个特性:

含头不含尾

简写:

1. 一直选取到结尾: 可省略第二个参数

2. 如果选取的元素离结尾近: 可用倒数下标:

arr.slice(arr.length-n,arr.length-m+1)

可简写为:arr.slice(-n,-m+1);

3. 复制数组:

arr.slice(0,arr.length);

可简写为:arr.slice();

2. 修改数组:splice 直接修改原数组

删除: arr.splice(starti,n);

删除arr中starti位置开始的n个元素

不考虑含头不含尾

其实: var deletes=arr.splice(starti,n);

返回值deletes保存了被删除的元素组成的临时数组

插入: arr.splice(starti,0,值1,值2,...)

在arr中starti位置，插入新值1,值2,...

原starti位置的值及其之后的值被向后顺移

强调: 不支持打散数组类型参数

替换: 其实就是删除旧的，插入新的

arr.splice(starti,n,值1,值2,...)

先删除arr中starti位置的n个值，再在starti位置插入新值

强调: 删除的元素个数和插入的新元素个数不必一致。

3.颠倒数组: arr.reverse();

强调: 仅负责原样颠倒数组，不负责排序

4.\*\*\*\*排序:将元素按从小到大的顺序重新排列

排序算法:(手写) 冒泡 快速 插入排序

冒泡:

正课:

1. \*\*\*数组:

2. \*\*\*String:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. \*\*\*\*排序:

手写排序算法: 冒泡，插入，快速

排序API:

arr.sort(): 默认将所有元素转为字符串再排列。

问题: 只能排列字符串类型的元素

解决: 使用自定义比较器函数

比较器函数: 专门比较任意两个数大小的函数

要求: 两个参数: a,b

返回值: a>b，就返回一个正数

a<b, 就返回一个负数

否则就返回0

比如:专门比较两个数字的比较器:

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

何时使用: 只要sort默认的排序规则不是想要的

就要自定义比较器函数，代替默认规则

如何使用: 将比较器函数作为参数传入sort方法中

arr.sort(compare);

强调: 传入sort方法内的比较器函数属于回调函数

回调函数: 将一个函数作为参数传入另一个函数中被反复调用

传入回调函数时，不要加()!

降序: 颠倒比较器函数的正负号

比如: function compare(a,b){return b-a;}

2. 栈和队列: js中没有专门的栈和队列类型，都是用数组模拟的

栈: 一端封闭，只能从一端进出的数组

特点: FILO

何时: 希望始终使用最新的元素时，就用栈结构

如何:

1. 结尾出入栈:

入栈: arr.push(值) ==> arr[arr.length]=值

出栈: var last=arr.pop()

特点: 每次出入栈都不影响现有元素的位置

2. 开头出入栈:

入栈: arr.unshift(值)

出栈: var first=arr.shift();

特点: 每次出入栈都会导致现有元素的位置发生顺移

队列: 只能从一端进入，从另一端出的数组

特点: FIFO

何时: 只要希望按照先来后到的顺序使用数组元素时

如何:

结尾入: arr.push(值)

开头出: var first=arr.shift();

3. 二维数组:

什么是: 数组中的元素又引用了另一个子数组

何时:

1. 如果要存储横行竖列的二维数据时

2. 如果要对数组中存储的数据再进行细致分类时

如何:

创建: 2种:

1. 先创建空数组，再向空数组中添加子数组

var data=[];

data[0]=[2,2,4,0];

data[1]=[2,0,2,2];

2. 在创建数组同时，初始化子数组

var data=[

[2,2,4,0],

[2,0,2,2],

];

访问二维数组中的元素: arr[r][c]

强调: 二维数组行下标不能越界: 会报错

二维数组的length属性: 仅表示子数组的个数

遍历二维数组: 外层循环遍历行，内层循环遍历列

for(var r=0;r<arr.length;r++){

for(var c=0;c<arr[r].length;c++){

arr[r][c]//当前元素

}

}

2. String: 其实就是一串字符组成的只读数组

vs 数组: 1. 下标

2. length

3. slice,concat

Day03

正课:

1. \*\*\*String

String API

\*\*\*\*\*正则

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

两个概念:

ECMAScript内置对象:ES标准中规定的，浏览器厂商已经实现的对象。

11个: String Number Boolean - 包装类型

Array Date Math RegExp

Error

Function Object

Global(全局对象)在浏览器中被window替代

包装类型: 专门封装原始类型的值，并提供对原始类型的值执行操作的API 的对象

为什么:

问题一: 原始类型的值本身不包含任何方法

解决: 预定义了三种包装类型，包装类型中包含操作数据的API。在试图对原始类型的值调用方法时，会自动创建对应包装类型的对象，封装原始类型的值，并调用对象的方法，操作值。

包装类型的对象，调用完方法后，自动释放！

问题二: string类型的数据，不能使用toFixed(2)

创建包装类型的对象前，先检查原始类型的值，再创建对应类型的包装类型对象。

不同类型的包装类型对象，拥有不同的API，不通用。

问题三: null和undefined不能打点?

鄙视题:

var str="Hello";//new String("Hello");

str.len=5;//为str添加一个属性len

console.log(str.len);//undefined

//希望len=5,始终留在str上?

String API: \*\*\*强调: 所有StringAPI都无权修改原字符串，只能返回新字符串

1. 大小写转换: 将字符串中所有英文字母统一转为大写或小写

str.toUpperCase();

str.toLowerCase();

何时: 只要希望不区分大小写时

如何: 今后只要不区分大小写，都要先统一转为大写/小写,再操作。

补: css中,text-transform: uppercase/lowercase/capitalize

2. 获取指定位置的字符:

str[i]

var char=str.charAt(i)

获得指定位置字符的unicode号

var unicode=str.charCodeAt(i);//省略i，默认为0

将unicode号反向转为字符:

String.fromCharCode(unicode);

3. 选取子字符串:

var subStr=str.slice(starti,endi+1);支持负数

str.substring(starti,endi+1);不支持负数

str.substr(starti,n);选取starti开始的n个

不考虑含头不含尾

正课:

1. \*\*\*String

StringAPI

2. \*\*\*\*\*正则表达式

StringAPI:

1. 检索关键词:

4种:

1. 查找一个固定关键词的出现位置:

var i=str.indexOf("关键词",fromi);

在str中从fromi位置开始查找下一个"关键词"所在的下标位置。

强调: 1. 返回下标位置，如果找不到，返回-1

2. 省略fromi，默认从0开始找

固定套路: 查找所有关键词的位置:

while((i=str.indexOf("关键词",i+1))!=-1){

i //本次找到的关键词位置

}

var i=str.lastIndexOf("关键词",fromi);

在str中从fromi位置开始查找前一个"关键词"所在的下标位置。

强调: 省略fromi，默认从最后(length-1)位置开始找

何时: 专门用于检索最后一个关键词的位置时

问题: 只能查找一个固定的关键词

解决: 用模式匹配——正则表达式

2.仅判断是否包含符合规则的关键词:

var i=str.search(/正则表达式/)

查找str中第一个符合正则表达式要求的关键词的位置i

如果没找到，返回-1

vs indexOf:

相同: 默认都是从0位置开始找第一个关键词的位置

如果找不到，都返回-1

不同:

indexOf: 优: 可指定开始位置！可查找所有

缺: 不支持模式匹配,每次只能找一个关键词

search: 缺: 不可指定开始位置，永远从0开始找

优: 支持模式匹配

3.列举所有关键词的内容:

var kwords=str.match(/正则表达式/g);

检索出str中所有和正则表达式匹配的关键词

返回值是包含所有关键词内容的数组

如果没找到，返回null

其中: g 表示查找所有，必须加

为什么: 正则默认仅匹配第一个关键词

优: 获得所有关键词的内容

缺: 无法获得每个关键词的位置

4.即获得每个关键词的内容，又获得每个关键词的位置:

RegExp对象?

\*\*\*\*\*正则表达式:

什么是: 规定一个字符串中字符出现规律的表达式

为什么: 现实中很多关键词的格式和内容都是有规律的，但又是不一样的。

何时: 1. 按规则，模式匹配多个关键词——检索

2. 按规则，检查一个字符串的格式——验证

如何:

1. 最简单的正则就是关键词原文

2. 字符集: 规定一位字符可用的备选字符列表

何时: 如果关键词的某一位字符有多种备选时

如何: [备选字符列表]

强调: 一个字符集[]，只能且必须匹配一位字符

简写: 如果字符集中的字符部分连续,可用-省略中间的字符

比如: 一位数字: [0-9]

一位小写字母: [a-z]

一位大写字母: [A-Z]

一位字母: [A-Za-z]

一位汉字: [\u4e00-\u9fa5]

除了...: [^47] 除了4和7之外都行！

强调: ^必须放在[]内部的开头！

3. 预定义字符集: 对常用字符集的最简化写法:

一位数字: \d

一位字母,数字或下划线: \w -> [A-Za-z0-9\_]

一位空字符: \s 包括: 空格，制表符

通配符: 匹配任意字符: .

强调: 只有在规则完全匹配时，才使用预定义字符集

只要有一点不匹配，都要手动编写普通字符集

4. 量词: 规定一个字符集出现次数的规则

何时: 只要规定一个字符集出现的次数

强调: 两次都要在字符集之后

有明确边界:

字符集{n,m} 至少n个，最多m个

字符集{n,} 至少n个，多了不限

字符集{n} 必须n个

没有明确边界:

字符集? 可有可无，最多1个

字符集+ 至少一个，多了不限

字符集\* 可有可无，多了不限——不限制个数

5. 选择和分组

分组: ()将多个字符集包裹为一组

为什么: 量词默认仅修饰相邻的前一个字符集

何时: 希望量词同时修饰多个字符集时，就要用()将多个字符集分为一组，用量词修饰分组。

选择: 或者 规则1|规则2 只要满足任意一个规则即可

比如: 滴滴出行: 防 微信 weixin wx

(微|w(ei)?)\s\*(信|x(in)?)

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

34578中选一个

9位数字

(\+86|0086)?\s\*1[34578]\d{9}

6.指定匹配位置:

^ 字符串开头: 比如: 开头的空字符: ^\s+

何时: 仅匹配开头位置的xxx

$ 字符串结尾: 比如: 结尾的空字符: \s+$

何时: 仅匹配结尾位置的xxx

同时匹配开头或结尾的空字符: ^\s+|\s+$

\b 单词边界: 字符串开头 字符串结尾 空字符 标点...

何时: 匹配单个单词时

7.密码强度: 6~8位 字母，数字的组合，至少包含一个大写字母和一个数字

预判: (?!规则$) 如果从头到尾不符合规则的要求

1.排除: 不全由小写字母和数字组成

说明: 可能有大写字母或特殊字符

(?![a-z0-9]+$)

2.排除: 不全由字母组成

说明: 可能有数字或特殊字符

(?![A-Za-z]+$)

3.只能由字母和数字组成

[A-Za-z0-9]{6,8}

最终: (?![a-z0-9]+$)(?![A-Za-z]+$)[A-Za-z0-9]{6,8}

正课:

1. \*\*\*String:

替换

切割

1. \*\*\*String:

所有String API无权修改原字符串，只能返回新字符串

替换

简单替换公式：str=str.replace(/正则/g,"替换值")

如果替换所有，就必须加g

如果英文环境下，忽略大小写，就必须加i

Ex1:

问题：挑出违禁字，将违禁字换成星

var str="No zuo no die,No can no bibi";

//str="we two who and who";

var kwords=str.match(/no/ig);

//null

str=str.replace(/no/ig,"\*\*");

console.log(str);

console.log("共替换"+ (kwords==null?0:kwords.length)+"处");

//用了一个三木运算 如果为空 则返回0，如果为数则返回length

高级替换: 根据关键词的不同，动态替换不同的新值

str=str.replace(/正则/g,function(kw,$1,$2…){

//kw获得本次找到的完整关键词

//$n会自动获得本次找到关键词中

return 根据kw的不同，返回不同的值})

Ex:

问题：挑出违禁字 二个字的换成两个星 三个字的换成三个星

//我去,我草,我去草,我草草,我去去

var str="那天，我去了她家，我说:'我草，你家真大'。她说，想和我去草地野餐。我草草的收拾了草地。她说:'我去去就来'";

str=str.replace(/(我)([去草]{1,2})/g,function(kw, $1, $2){

//我去 我 去

//我去草 我 去草

//kw会自动获得每次匹配的关键词

return $1+($2.length==1?"\*":"\*\*");

}

//返回值会替换到当前关键词的位置

console.log(str);

Ex:

问题：将每个单词首字母改成大写

var str="you can you up";

//用正则找到每个单词,将单词分为首字母和剩余字母两组

//替换时，替换函数返回首字母转大写，再拼上剩余字母

str=str.replace(/\b[A-Za-z]/g,function(kw){

return kw.toUpperCase();

}

练习:

如果需要对关键词部分处理或分组处理: 2步:

1. 先用()，将正则分组

匹配时，每个()，会自动获得一个分组编号:1,2,3...

2. 在替换时，使用$1,$2,代替每个分组本次匹配的子内容

str=str.replace(/正则/g,function(kw,$1,$2,...){

//kw获得本次找到的完整关键词

//$n获得本次找到的关键词中第n个分组的子内容

return 返回不同的值

});

衍生:

删除: 替换成空字符串

ex:

<script type="text/javascript">

var str=" \tzhang dong\t ";

function ltrim(str){//删除str开头的空字符

return str.replace(/^\s+/,"");

}

function rtrim(str){//删除str结尾的空字符

return str.replace(/\s+$/,"");

}

function trim(str){//删除str开头和结尾空字符

return str.replace(/^\s+|\s+$/g,"");

}console.log(ltrim(str));//"zhang dong\t "

console.log(rtrim(str));//" \tzhang dong"

console.log(trim(str));//"zhang dong"

格式化: 2步:

1. 用正则对原始字符串分组

2. 在replace的替换值中使用$n重新拼接新格式

简写：birth.replace(/(\d{4})(\d{2})(\d{2})/,"$1年$2月$3日");

var birth="19831226";

birth=birth.replace(

/(\d{4})(\d{2})(\d{2})/,"$1年$2月$3日");

console.log(birth);

切割

定义：按指定分隔符，将字符串切割为多段子字符串

简单切割: 分隔符是固定: var subs=str.split("分隔符")

Ex:

//将邮箱切割成两段

var mail="zhangdong@tedu.cn";

var subs=mail.split("@");

//[zhangdong,tedu.cn]

var uname=subs[0];

var domain=subs[1];

console.log(uname,domain);

固定套路:

将字符串打散成字符数组:var chars=str.split("");

ex:

var str="Hello";

//将hello切割成单个字母

console.log(str.split(""));

//返回一个数组

复杂切割: 分隔符不是固定: var subs=str.split(/正则表达式/)

ex:

//获得ul的内容

var html=ulNames.innerHTML;

//删除html开头的空字符和<li>

//删除html结尾的</li>和空字符

html=html.replace(

/^\s\*<li>|<\/li>\s\*$/ig,"");

console.log(html);

//将html按</li>空字符<li>切割为数组,保存在names中

var names=html.split(/<\/li>\s\*<li>/);

console.log(names);

//将names按字符串升序排列

names.sort();

//将names拼接为li，将拼接的结果赋值给ulNames的innerHTML

ulNames.innerHTML=

"<li>"+names.join("</li><li>")+"</li>";

正课:

1. \*\*\*RegExp:

reg.exec(str)

reg.test(str)

2. Math

3. Date

1. \*\*\*RegExp对象:

什么是: 封装一条正则表达式,提供了使用正则执行查找和验证的API

查找

何时: 只要使用正则表达式，都要先创建正则表达式对象

创建: 2种:

1. 直接量: var reg=/正则表达式/ig;

使用场合：如果正则表达式一旦定义不再改变

字符冲突: 正则表达式中的/，都要转义为\/

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

使用场合： 如果正则表达式本身都需要动态生成

字符冲突: 正则表达式中的\和'"都要转义

比如: "\\d \\s \\w \" \'"

var names=["锄禾","白鹭"];

//锄禾|白鹭|日日|明月

var str="锄禾日当午，一行白鹭上青天，床前明月光，日日思君不见君";

//根据数组内容动态生成正则

var reg=new RegExp(names.join("|"),"g");

console.log(str.match(reg));

API:

1. 高级查找: 即查找每个关键词内容，又查找位置

var arr=reg.exec(str);

在str中查找下一个符合reg规则的关键词的内容和位置

固定套路: 查找每个关键词的内容和位置

var arr=null;

while((arr=reg.exec(str))!=null){

arr: ["关键词",$1,$2,... index:位置]}

0

原理: reg对象中有一个lastIndex=0属性,规定了exec下次开始查找的位置——不用手动维护

exec每执行一次，做三件事:

1.将本次关键词内容，放入数组第0个元素，将每个分组的字内容放入后续内容

2.将关键词位置，放入数组的index属性

3.自动将reg的lastIndex+=关键词的长度

导致下次自动跳过关键词，找下一个

强调: 也是一个一个找，每找一个新的，都会替换数组中旧的关键词。

简写: 如果只获得分组子内容，不需要完整关键词

RegExp.$n -> 可获得本次查找到的第n个分组的子内容

while(reg.exec(str)!=null){

RegExp.$n第N个分组的子内容

}

//查找每个关键词的内容和位置

var str="No zuo no die,No can no bibi";

var reg=/no/ig;

var arr=null;

while((arr=reg.exec(str))!=null){

console.log("在位置"+arr.index+"发现敏感词"+arr[0]);

console.log("下次从"+reg.lastIndex+"开始");

}

练习: 贪婪模式: 默认情况下，正则表达式都会匹配最长的符合条件的子字符串

比如: .\* .+ ...

懒惰模式: 让正则表达式仅匹配最短的符合条件的子内容

.\*? .+?

Ex:

Var html='<a name="#top"></a><h1>Welcome</h1><ul><li><a class="curr" href="http://tmooc.cn">go to tmooc</a></li><li><a href="http://tedu.cn" target="\_blank">go to tedu</a></li></ul>';

var reg=/<a\s+[^>]\*?href="([^"]\*?)"/ig;

while(reg.exec(html)!=null){

console.log(RegExp.$1);

}

2.验证: var bool=reg.test(str);

验证str是否符合reg的要求。

问题: test方法，默认只要部分匹配即可

解决: 只要验证，必须前加^后加$，要求从头到尾完整匹配

var reg=/^\d{6}$/;

///^(?![a-z0-9]+$)(?![A-Za-z]+$)[A-Za-z0-9]{6,8}$/

while(!reg.test(prompt("输入密码"))){

alert("密码格式不符,请重新输入");

}

document.write("<h1 style='color:green'>验证通过</h1>");

2. Math:

什么是: 专门封装数学计算的常量和API的对象

不能创建new！

API: 所有API必须用Math.直接调用

1. 取整:

Math.ceil(num) 上取整: 只要超过，就取下一个整数

Math.floor(num) 下取整: 只要超过，就抹掉小数部分

vs parseInt(str):

floor: 参数是number,

不能去掉字符串结尾的非数字字符

parseInt: 参数是字符串，

专门去掉字符串结尾的非数字字符

在省略小数部分

缺: 强行去掉小数部分，会导致误差

多数情况只去单位，还要保留小数

解决: 如果只是去单位 首选parseFloat

Math.round(num) 四舍五入取整:

vs toFixed(d):

round: 必须用Math直接调用

缺: 只能取整，不能指定小数位数

优: 返回值: number 可直接算数计算

Num.toFixed(d): 可被任意数字类型的值调用

优: 可以按任意小数位数四舍五入(0<=d<=20)

缺: 返回值: string 先转为number再计算

解决：自定义round方法: 可按任意小数位数四舍五入

返回number

Function round(num,d){

Num\*=math

}

2. 乘方和开平方:

乘方: Math.pow(底数,幂)，比如: Math.pow(10,2)=100

开平方: Math.sqrt(num)

for(var i=2;i<Math.sqrt(num);i++){

if(num%i==0){不是质数;break}

}

是质数

Ex:

function round(num,d){

//num\* 10的d次方

num\*=Math.pow(10,d);

//对num四舍五入取整

num=Math.round(num);

//num/ 10的d次方

num/=Math.pow(10,d);

return num;

}

//测试:

console.log(

round(345.678,2),//345.68

round(345.678,0),//346

round(345.678,-1)//350

);

3. 最大值和最小值:

Math.max(值1,值2,...)

Math.min(值1,值2,...)

不支持获得数组中的最大值和最小值

变通实现:

Math.max.apply(null,arr);//获得arr中的最大值

apply也具有打散数组参数的能力

Math.min.apply(null,arr);

4. 随机数:

0<=Math.random()<1

任意max和min之间取随机整数的公式:

var r=Math.floor(Math.random()\*(max-min+1)+min)

在0~max之间取随机整数:

var r=Math.floor(Math.random()\*(max+1))

ex:

function doubleball(){

var reds=[];//定义空数组reds

//反复: 只要reds中少于6个

while(reds.length<6){

//在1~33之间生成一个随机整数r

var r=Math.floor(Math.random()\*33+1);

//遍历reds中每个元素

for(var i=0;i<reds.length;i++){

//如果reds中当前元素等于r，就退出循环

if(reds[i]==r){break;}

}//(遍历结束)

//如果i等于reds的元素个数,就将r压入reds中

if(i==reds.length){reds.push(r);}

}//(循环结束)

//将reds按数字升序排列

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

//在1~16之间生成一个随机整数blue

var blue=Math.floor(Math.random()\*16+1);

//返回 reds转为字符串 拼个| 拼上blue

return String(reds)+"|"+blue;

}

document.write("<h1>"+doubleball()+"</h1>");

3. Date:

什么是: 封装一个日期,提供对日期执行操作的API

何时: 今后只要保存日期或操作日期，都要使用Date对象

如何:

创建: 3种情况:

1. 创建日期对象，并自动获得客户端当前系统时间:

var now=new Date();

2. 创建日期对象，保存自定义时间:

var date=new Date("yyyy/MM/dd hh:mm:ss");时分秒可以省略

new Date(yyyy,MM-1,dd,hh,mm,ss);

3. 复制一个日期对象:

问题: 日期对象的计算都是直接修改原日期对象

修改后，无法保留旧的时间

解决: 在计算之前，如果需要保存计算前后的时间 先将旧时间复制一个副本另存。

var date1=new Date(.....);

var date2=new Date(date1);//将date1复制一个副本

4.

日期对象的原理:

日期对象中封装的其实是一个1970年1月1日0点至今的毫秒数 毫秒数不会用完

Ex:

var now=new Date();

console.log(now);

console.log(now.getTime());

console.log(Number.MAX\_VALUE);

var maxYears=

Number.MAX\_VALUE/1000/3600/24/365;

console.log(maxYears);

var birth=new Date("1983/12/26");

var days=(now-birth)/1000/3600/24;

console.log(days);

var org=new Date(0);

console.log(org.toGMTString());

正课:

1. Date

2. \*\*\*Error

1. Date:

什么是: 封装一个日期，提供对日期执行操作的API

何时: 今后只要保存或操作日期都用Date对象

如何:

创建: 4种:

1. 获得客户端当前系统时间:

var now=new Date();

2. 创建日期对象保存自定义时间:

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

new Date(yyyy,MM-1,dd,hh,mm,ss)

3. 复制一个日期:

为什么: 日期的计算都直接修改原日期对象

旧时间无法保存

何时: 如果需要同时保存计算前后的两个时间时，就必须将原日期复制一个副本再计算。

如何: var date2=new Date(date1);

4. 直接用毫秒数创建日期对象:

var date=new Date(ms)

原理: Date对象中保存的是1970年1月1日0点至今的毫秒数

Var ms=date.getTime(ms)

API:

单位: 年FullYear 月Month 日Date 星期Day

时Hours 分Minutes 秒Seconds 毫秒MilliSeconds

1. 每个单位都有一对儿getXXX/setXXX方法

getXXX()方法负责读取指定分量的值:

比如: date.getDate(); //获得几号

setXXX(n) 方法负责修改指定分量的值:

比如: date.setDate(10); //修改date的日期为10号

特例: Day没有set方法，因为星期只读，不可改

2. 取值范围:

(1)只有月中的日(Date)从1开始到31结束

(2)其余单位都是从0开始，到进制-1结束

只有Month需要修正 0~11 计算机中的月份比现实中都小1

其实都不需要修正

Date 1~31 不用修正

Day 0~6 星期日是0，不需要修正

Hours 0~23 不用修正

Minutes,Seconds 0~59 不用修正

日期计算:

1. 两日期相减: 获得毫秒差（非常有用）

2. 对任意单位做加减: 3步:

1. 取分量: var n=date.get分量();

2. 做加减: n+=?;

3. 放回去; date.set分量(n);

特点: 1. 自动调整进制

2. 直接修改原日期对象 同时保证新旧两个时间 要存副本

简化版: date.set分量(date.get分量()+n)

date.setDate(date.getDate()+3)

格式化--转字符串:

.toString() -> 转为中国标准时间(+8区)

.toLocaleString() -> 转为系统当前时间格式

.toLocaleDateString -> 转为系统当前时间格式

仅包含日期部分但是有兼容性问题

.toLocaleTimeString -> 转为系统当前时间格式

仅包含时间部分

.toGMTString() -> 转为国际标准时间(0时区)

EX:

var now=new Date();//获得当前时间

var end=new Date(now);//复制时间副本

end.setDate(end.getDate()+59);//用副本做计算

//同时显示开始和结束时间

console.log(now+" - "+end);

ex:

//定义入职时间hiredate: 2012/6/30

var hiredate=new Date("2012/6/30");

//复制入职时间为到期时间enddate

var enddate=new Date(hiredate);

//将enddate+3年

enddate.setFullYear(enddate.getFullYear()+3);

//复制到期时间为续签时间renewdate

var renewdate=new Date(enddate);

//将renewdate-1月

renewdate.setMonth(renewdate.getMonth()-1);

//如果renewdate是周六:

if(renewdate.getDay()==6){

//renewdate-1天

renewdate.setDate(renewdate.getDate()-1);

}else if(renewdate.getDay()==0){

//否则，如果renewdate是周日:

//renewdate-2天

renewdate.setDate(renewdate.getDate()-2);

}

//复制续签时间为提醒时间alertdate

var alertdate=new Date(renewdate);

//将alertdate-7天

alertdate.setDate(alertdate.getDate()-7);

console.log("到期时间"+enddate);

console.log("续签时间"+renewdate);

console.log("提醒时间"+alertdate);

ex:

var now=new Date();

var days=10;//10个工作日之后

//i从0开始，到<days结束

for(var i=0;i<days;i++){

//如果now是周五

//now+3天

//否则，如果now是周六

//now+2天

//否则

//now+1天

var day=now.getDay();//获得now的星期

now.setDate(

now.getDate()+(day==5?3:day==6?2:1)

);

}

console.log(now);

ex:

问题：var date=new Date("2016/9/10");

console.log(date.toLocaleDateString());

以上方法不适用于每个浏览器

为了解决浏览器兼容问题 自定义一套函数适用于每个浏览器

function format(date){

//定义数组week: 包含日,一,二,....,六

var week=

["日","一","二","三","四","五","六"];

//获得date中的年保存在y中

var y=date.getFullYear();

//获得date中的月+1后保存在M中

var M=date.getMonth()+1;

M<10&&(M="0"+M);//如果M<10，改为0+M

//获得date中的日保存在d中

var d=date.getDate();

d<10&&(d="0"+d);//如果d<10，改为0+d

//获得date中的星期保存在day中

var day=date.getDay();

//获取week中day位置的汉字保存回day中

day=week[day];

//获得date中的小时保存在h中

var h=date.getHours();

//声明变量am，如果h<12，am就赋值为上午,否则赋值为下午

var am=h<12?"上午":"下午";

h>12&&(h-=12);//如果h>12,就h-12

h<10&&(h="0"+h);//如果h<10,就改为0+h

//获得date中的分钟保存在m中

var m=date.getMinutes();

m<10&&(m="0"+m);//如果m<10,就改为0+m

return y+"年"+M+"月"+d+"日 星期"+day

+" "+am+h+":"+m;

}//2016年10月10日 星期一 上午11:28

console.log(format(new Date()));

正课:

1. \*\*\*Error

2. \*\*\*\*\*Function

1. \*\*\*Error:

错误: 在程序执行过程中，导致程序异常中断的特殊情况

错误处理: 在程序执行过程中，即使程序出错，也能保证不会异常中断的机制。

如何使用:

try{

可能出现错误的语句

}catch(err){//err: 自动获得错误对象——不可省略

一旦出错，才执行的错误处理代码: 提示错误, 记录日志

}finally{

无论是否出错，都必须执行的代码: 释放资源

}

err错误对象

Ex:

var num=parseFloat(

//prompt("输入数字")//请用户输入num

//);

//var d=prompt("输入小数位数");//请用户输入d

try{

console.log(num.toFixed(d));

}catch(err){

//catch中的代码只有出错才执行，不出错不执行

//err: 自动获得错误对象，包含错误信息

console.log(String(err));

}finally{//无论是否出错，都必须执行

console.log("释放资源");

num=null; d=null;

}

if(!isNaN(d)&&d>=0&&d<=20){

console.log(num.toFixed(d));

}else{

console.log("参数超范围: 小数位数必须介于0~20之间");

}

//finally

console.log("释放资源"); num=null; d=null;

console.log("正常结束");

try{

new XMLHttpRequest();//尝试创建XHR类型对象

document.write("您的浏览器完美支持Ajax");

}catch(err){//如果不能创建，说明不支持

document.write("您的浏览器不支持Ajax技术，请下载最新版本");

}

//所有能new的对象，已经包含在window下,不是undefined的说明可以用

if(window.XMLHttpRequest!==undefined){

document.write("您的浏览器完美支持Ajax");

}else{//是undefined的说明不包含，不能用

document.write("您的浏览器不支持Ajax技术，请下载最新版本");

}

Error: 封装错误信息的对象,在发生错误时，自动创建

err.name: 错误的名称

err.message: 错误的提示信息

string（err）：错误类型：错误提示信息

js中包含6种错误:

1.SyntaxError: 语法错误

2. ReferenceError: 引用错误 要用的东西没找到

3.RangeError: 范围错误 专指参数超范围

4. TypeError: 类型错误 错误的调用了对象的方法

5. EvalError,

6. URIError

优化：

原则: 1. try中尽量少的包含语句

2. 如果已经提前预知错误的类型，就用if...else...来代替try...catch

何时使用: 今后只要处理无法提前预料的错误采用try...catch

try...catch也用于处理浏览器兼容性问题:

比如: Ajax : XMLHttpRequest

抛出自定义错误:

throw new Error("自定义错误信息") Error包含六种错误

作者与调用者的交流

Ex:

//函数作者:

function round(num,d){

if(!isNaN(num)&&!isNaN(d)){

num\*=Math.pow(10,d);

num=Math.round(num);

num/=Math.pow(10,d);

return num;

}else{

throw new Error("孙子！参数必须是数字!");

}

}

//调用者:

var num=prompt("请输入一个数字");

var d=prompt("请输入小数位数");

try{

console.log(round(num,d));

}catch(err){

alert(err);

}

2. \*\*\*\*\*Function:

函数对象:

专门封装一个函数定义的对象

其实, js中一切函数都是引用类型的对象

创建: 3种

1. 以声明方式:

function 函数名(参数列表){函数体; return 返回值}

会被声明提前！

何时: 首选

声明提前(hoist):

定义：在程序开始执行前，先将所有var声明的变量和function声明的函数提前到当前作用域的顶部，集中创建

赋值留在原地

Ex:

（笔试题，会考）

原顺序 function fun(){console.log(1)}

fun();//2

function fun(){console.log(2)}

fun();//2

var fun=100;

console.log(fun);//100

fun();//报错

为什么会这样呢？

系统运行顺序：function fun(){console.log(1)}

function fun(){console.log(2)}

var fun;声明提前 打酱油的 只有=才有资格改变

fun();//2

fun();//2

fun=100; 赋值留在原地

console.log(fun);//100

fun();//报错

2. 函数直接量:

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

不会被声明提前

何时使用: 不希望被声明提前时

3. 用new:

var 函数名=new Function("参数1","参数2",...,"函数体")

何时: 参数或函数体可能动态生成时

注意：一般是底层框架使用

鄙视题: function compare(a,b){return a-b;}

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

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

ex:

var arr=[3,1,12,123,23,2];

/三种任选其一

//function compare(a,b){return a-b;}

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

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

//arr.sort(compare);

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

console.log(String(arr));

重载(overload):

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执行。

为什么: 减少API的数量，减轻调用者的负担

何时使用: 今后一项任务，只是根据传入参数的不同，执行不同的操作时，就用重载。

如何使用:

问题: js的语法不支持重载,因为js中不允许同时存在多个同名函数。

解决: arguments（类数组对象）: 每个函数内自动创建的

自动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. 下标访问每个元素

2. length属性记录元素个数

3. 用for 遍历

不同: 1.类型不同

2.不能使用数组API

Ex:

function pay(){//现金结账

//如果类数组对象的长度等于1，给一个参数

if(arguments.length==1){

//就现金结账

console.log("现金结账");

//如果类数组对象的长度等于2，给两个参数

}else if(arguments.length==2){

//就刷卡结账

console.log("刷卡结账");

}else{

//否则就微信，不给参数

console.log("微信结账");

}

}

pay(100);//现金结账

pay("xxxx xxxx","\*\*\*\*\*\*");//刷卡结账

pay();//微信结账

动态生成几个元素相加的值

function add(){

//遍历arguments中每个元素,同时声明sum=0将arguments中当前元素累加到sum上，再返回

for(var i=0,sum=0;i<arguments.length;sum+=arguments[i++]);

return sum;

}

console.log(

add(1,2,3),//6

add(1,2,3,4,5)//15

);

匿名函数: （重点）

什么是: 函数创建时没有被任何变量引用的

为什么: 节约内存

何时使用: 如果一个函数只用一次，不会重用（就像一次性饭盒一样，吃完不会留着，也不会在上面写名字）

1. 所有回调函数，几乎都要用匿名

2. 匿名函数自调: 创建一个函数，自己调用自己，调用后立刻释放。

3.公式：（function函数）（参数列表）

4.主要目的: 为了定义一个临时函数作用域，避免全局污染

Ex:

回调

var arr=[3,1,12,123,23,2];

/三种任选其一

//function compare(a,b){return a-b;}

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

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

//arr.sort(compare);

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

console.log(String(arr));

自调

<script>

//提示 在? 开始加载页面...

(function(){

var start=new Date();

alert("开始加载页面..+start.toLocaleTimeString());})();

</script>

<h1>Welcome</h1>

<ul>

<li>旭旭</li><!--Ctrl+Alt+↓-->

<li>东东</li>

<li>华华</li>

<li>黑胖子</li>

</ul>

<script>

//提示 在? 加载完成!

(function(){

var end=new Date();

alert("加载完成!"+end.toLocaleTimeString());})();

</script>

作用域(scope):

定义：一个变量的使用范围

其实，作用域是一个对象，所有变量都保存在各自的作用域对象中

函数的生命周期:

1.程序开始执行时:

创建执行环境栈(ECS):

依次保存正在调用的函数的执行环境 的数组

执行浏览器主程序(main):

创建window对象作为全局作用域对象

2.定义函数时:

在window中添加全局变量，函数名为变量名

创建函数对象，封装函数的定义

函数名变量引用函数对象

函数对象中有一个scope属性引用函数创建时的作用域对象

3. 调用函数时:

在ECS中压入正在调用的函数的执行环境

创建本次函数调用使用的函数作用域对象(活动对象AO)

本次函数调用的EC引用着AO

在AO中定义parent属性，引用函数的scope指定的作用域对象。

在AO中添加局部变量的定义

变量使用的顺序:

先用局部的，如果没有，才去全局找

4.函数调用后:

将本次函数调用的EC出栈！

导致本次函数调用的函数作用域(AO)对象释放

导致局部变量一同释放

作用域链(scope chain):

在函数调用时

由各级作用域对象逐级引用形成的链式结构

作用域链控制着变量的使用顺序

先用局部的，如果没有，才去全局找

<script>

var a=10;

function fun(){

var a=100;

a++;

console.log(a);

}

fun();//101

console.log(a);//10

</script>

正课:

1. \*\*\*\*\*Function

闭包

2. \*\*\*\*\*面向对象

1. \*\*\*\*\*Function

闭包:

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: 优: 可反复使用

缺: 易被污染——尽量不使用全局变量

局部变量: 缺: 无法反复使用

优: 仅函数内可用，不会被污染！

何时: 只要希望重用一个变量，又不希望变量被污染时

如何: 三特点:

1. 用外层函数包裹受保护的变量和操作变量的内层函数

2. 外层函数返回内层函数对象

3. 使用闭包的人，调用外层函数，获得内层函数对象

鄙视: 2步: 判断闭包输出

1. 找受保护的变量，判断在外层函数调用后，受保护的变量的值。

2. 找到所有操作受保护变量的内层函数:

特殊: 如果闭包只希望创建一次，反复使用，外层函数就要使用匿名函数自调。

var fun=(function(){var 变量return function(){

...变量...}})();

//fun->function(){ ...变量... }

闭包的缺点: 比普通的函数占用更多内存

（外层函数AO始终不释放）

容易内存泄漏

如何释放闭包: 设置引用内层函数的变量为null

导致内层函数对象释放

导致外层函数的AO释放

闭包就释放了

Ex:

**//Step1: 用外层函数包裹受保护的变量和操作变量的内层函数**

**function outer(){**

**//受保护的变量(地下党)**

**var i=1;**

**//Step2: 外层函数返回内层函数对象（线人）**

**return function(){**

**console.log(i++);**

**}**

**}**

**//Step3: 使用者调用外层函数，获得内层函数对象**

**var next=outer();**

**//按理说: outer调用后，i应该被释放**

**//为什么还能重用？！因为他们形成了闭包**

**//next: function(){(i=1) console.log(i++); }**

**next();//1**

**next();//2**

**i=0;**

**next();//3**

**next();//4**

**next=null;//释放闭包**

function fun(){

var n=999;

nAdd=function(){n++;}

return function(){

console.log(n);

}

}

var get=fun();

get();//999

nAdd();

get();//1000

**function fn(){**

**for(var i=0,arr=[];i<3;i++){**

**arr[i]=function(){console.log(i);}**

**}**

**return arr;**

**}//i=3**

**var funs=fn();**

**//funs:arr:[**

**//0: function(){console.log(i);}**

**//1: function(){console.log(i);}**

**//2: function(){console.log(i);}**

**//]**

**funs[0]();//3**

**funs[1]();//3**

**funs[2]();//3**

ex:

此闭包不能使用简写 因为有参数

function factory(rate){//生产计算器的工厂

return function(rmb){//要生产的计算器

return (rmb\*rate).toFixed(2);

}

}

var rmb2$=factory(0.1639);

//rmb2$: function(rmb){(rate=0.1639)

//return (rmb\*rate).toFixed(2);

//}

console.log(rmb2$(1000));//163.9

var rmb2Euro=factory(0.1449);

console.log(rmb2Euro(1000));//144.9

2. \*\*\*\*\*面向对象:

什么是面向对象: 程序中都是先用对象来描述现实中一个事物

再根据需要调用对象的方法执行操作

为什么: 和现实中的做事方法几乎完全一样。

什么是对象:

本质： 内存中集中存储多个数据和方法(函数)的存储空间, 起一个名字。

用途: 程序中描述现实中一个具体事物的程序结构

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(函数)

如何:

创建对象: 也叫封装: 将一个事物的属性和功能集中定义在一个对象中。

3种:

1. 对象直接量:

var obj={

属性名: 属性值,

... : ...,

方法名: function(){

}

};

函数 vs 方法: 所有function都是函数

保存到对象中的function也称为方法

访问对象的成员: 成员: 属性和方法

访问属性: 对象.属性名 -> 用法和普通变量完全一样！

的

调用方法: 对象.方法名() -> 用法和普通方法完全一样!

其实方法也是属性，只不过存储的是一个函数而已。

Ex:

//创建对象lilei

//name: Li Lei

//age: 11

//intr: 说: "I'm Li Lei, I'm 11"

var lilei={

sname:"Li Lei",

sage:11,

intr: function(){

console.log("I'm "+sname+", I'm "+sage);

}

}

console.log(lilei.sage);//11

lilei.sage++;

console.log(lilei.sage);//12

lilei.intr();

正课:

1. \*\*\*\*\*OOP

封装

继承

1. 封装: 创建对象: 3种:

1. 对象直接量:

var obj={

属性名:属性值,

... : ... ,

方法名: function(){

...this.属性名...

}

}

问题: 对象自己的方法，如何访问当前对象自己的属性呢？

错误: 直接写属性名: 报错: 变量名未定义

解决: 任何对象自己的方法，想使用自己的属性，都必须:

this.属性名 其中: this.翻译为: 当前对象的xxx

\*\*\*\*\*this: 在方法调用时，自动获得正在调用方法的.前的对象

比如: lilei.intr() : intr中的this->lilei

原理: this在函数调用时，自动存储在函数作用域对象(AO)中

被自动设置为引用当前函数.前的对象

方法中: 不加this.的变量，默认仅在作用域链中找

加this.的属性，被引导到当前对象中找属性，不再查找作用域链。

2. 用new: 2步:

var obj=new Object(); new可省, ()可省, 不能同时省略

obj.属性名=属性值;

obj.方法名=function(){ ...this.属性名... };

何时: 在创建对象时，暂时不知道对象的成员

揭示: js中一切对象底层都是关联数组

对象的属性名其实是关联数组的key

对象的属性值其实是关联数组的value

后果: 对象的成员: 对象["属性名"]

对象.属性名 <==> 对象["属性名"]

对象.key 对象["key"]

对象可随时添加新属性

用for in遍历对象的每个属性

问题: 以上两种方法只适合创建一个对象

如果反复创建多个相同结构，但值不同个对象，会导致大量重复代码！

解决: 先定义对象的结构，再用相同结构反复创建对象

3. 构造函数(constructor):

什么是构造函数: 描述一类对象统一结构的特殊函数

何时: 只要反复创建多个结构相同，但值不同的一类对象时，都要先定义构造函数，再用构造函数反复创建对象

如何使用: 2步:

1. 定义构造函数:

function 类型名(属性参数){

this.属性名=属性参数;//向新对象中添加新属性

this.方法名=function(){//向新对象中添加新方法

... this.属性名 ...

}

}

2. 用new调用构造函数，传入属性值作为参数，获得新对象

var obj=new 类型名(属性值);

new: 4件事:

1. 创建空对象: new Object()

2. 让新对象继承构造函数的prototype

3. 用新对象调用构造函数: 将构造函数中的this自动换成新对象

4. 返回新对象的地址

优: 代码重用！

问题: 对象的方法被反复创建了多次，缺: 浪费内存

解决: 继承:

2. 继承:

什么是: 父对象中的成员，子对象不必反复创建，可直接使用

为什么: 即代码重用，又节约内存

何时: 只要多个子对象，希望拥有相同的成员时

就要将相同的成员集中保存在父对象中一份即可！

如何:

原型对象(prototype): 集中存储一类子对象共有成员的父对象

访问原型对象:

定义构造函数时，都会自动定义一个该构造函数的原型对象。

构造函数中有一个prototype属性指向当前类型的原型对象

向原型对象中添加共有成员:

构造函数.prototype.方法=function(){ ... }

正课:

1. \*\*\*\*\*OOP

封装

继承

多态

面向对象三大特点: 封装，继承，多态

封装: 将一个事物的属性和功能集中定义在一个对象中

继承: 父对象中的成员，子对象可以直接访问

多态: 同一个事物，在不同情况下表现出不同的状态

继承:

js中的继承，都是通过原型对象实现的

原型对象: 集中存储一类对象的共有成员的父对象

访问: 构造函数.prototype

向原型对象中添加成员: 构造函数.prototype.成员=function

构造函数创建出的子对象，都有\_\_proto\_\_属性继承构造函数的原型对象prototype

自有属性和共有属性:

自有属性: 直接保存在子对象本地的属性都是自有属性

共有属性: 保存在父级原型对象中，被所有子对象共用的属性

为对象扩展属性:

为单个对象扩展自有属性: 对象.属性名=值

为一类对象扩展共有属性: 构造函数.prototype.属性名=值

修改属性:

修改自有属性: 对象.属性名=值

修改共有属性: 必须用 构造函数.prototype.属性名=值

删除属性:

删除自有属性: delete 对象.属性名;

删除共有属性: 必须用 delete 构造函数.prototype.属性名

判断自有还是共有:

判断自有属性: var bool=obj.hasOwnProperty("属性名")

判断obj的本地是否包含自有属性"属性名"

判断共有属性: !obj.hasOwnProperty("属性名")

&&obj.属性名!==undefined

内置对象的原型对象:

new Array() String() RegExp() Date()

Error() Function() Object()

一切都是构造函数

每个类型都对应自己的prototype对象

prototype对象中集中存储着该类型可用的所有API

原型链(prototype chain): 由各级父元素连续继承，形成的链式结构。

控制着: 对象.属性的使用顺序

先用自有的，如果没有，才沿原型链向上找

vs 作用域链: 控制着不带.的普通变量的使用范围

先用局部，再用全局

鄙视题: 判断一个对象是不是数组类型: 共几种方法

错误: typeof()——只能区分原始类型和function，无法区分内置对象类型

解决:

1. 判断原型对象: var bool=father.isPrototypeOf(child)

判断father是不是child的父对象

2. 判断构造函数:

var bool=child instanceof 构造函数

判断child是不是用指定的构造函数创建出来的

instance: 实例: 用构造函数创建出的一个子对象

实例化: 用new调用构造函数创建一个对象

问题: 前两个检查不仅检查直接父对象，而且还检查整个原型链。

解决:

3. 检查对象的class属性: 和原型链无关

每个对象中都有一个内部属性class: 记录了对象创建时的类型。一旦创建，不会更改。

问题1: class不允许用.直接访问

解决: obj.toString() -> "[object class]"

其中class可能为: Object Array Date ...

问题2: 多数内置对象重写了toString方法

重写的toString方法已经无法返回class

解决: 临时借用函数:

函数.call(对象) -> 对象.函数()

Object.prototype.toString.call(obj1)

-> obj1.toString();

4. ES5 : Array.isArray(obj)

多态: 同一个方法在不同情况下表现出不同状态

重写: 子对象觉得父对象的成员不好用

可在本地定义同名自有属性，覆盖父对象成员

使用时，优先使用子对象的成员。

为什么: 体现子对象和父对象之间的差异

自定义继承:

1. 修改子对象的\_\_proto\_\_指向指定的父对象:

child.\_\_proto\_\_=father;

问题: \_\_proto\_\_是内部属性，可能被禁用:

解决: 获得父对象: Object.getPrototypeOf(child)

获得child的原型父对象

设置父对象: Object.setPrototypeOf(child, father)

设置child继承father

何时: 如果只修改两个对象间的继承

2. 通过修改构造函数的prototype属性，同时修改所有子对象的父对象:

构造函数.prototype=father

强调: 时机: 一定要在开始创建对象前就换

3. 两种类型间的继承:

何时: 只要发现两个类型间，拥有相同的属性结构和方法定义

如何:

1. 定义抽象父类型，保存子类型共有的属性结构和方法定义

父类型的构造函数中保存： 共有的属性结构

父类型的原型对象中保存 : 共有的方法定义

2. 在子类型构造函数中调用父类型构造函数

问题: 不能直接调用

如果直接调用父类型构造函数中的this->window

解决: call: 2功能: 1. 借用本来不能调动的函数

2. 替换函数中的this为想要的对象

3. 让子类型原型对象，继承父类型原型对象

Day08

ECMAScript JavaScript JScript

ECMAScript: ECMA指定的JavaScript语言标准——规定了核心语法

JavaScript: NetScape参照ECMAScript标准，实现的JavaScript版本

JScript: MS参照ECMAScript标准，实现的JavaScript版本

1. 对象的属性:

命名属性: 可直接用.访问的属性

(1)数据属性: 直接存储一个属性值的属性

四大特性: { //对属性提供基本的保护

value: //实际存储属性值,

writable: true/false ,//控制属性是否可修改

enumerable: true ,//控制能否在for in时遍历到

　　　　　　　　　　　 //即使为false，也能用.访问到

configurable: true,false//1. 控制能否删除

//2. 控制能否修改以上两个特性

//属性值value依然可以修改

}

如何获取属性的四大特性:

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

attrs:{value: xxx, writable: xxx,...}

如何修改属性的特性:

Object.defineProperty(obj,"属性名",{

特性名: 值,

configurable:false

... : ...

})

问题: 别人也可能将特性的值改回true

解决: 只要修改特性，都要加上configurable:false

来进制修改其他特性的值

configurable:false:　不可逆！

问题：Object.defineProperty(obj,"属性名"怎么添加新属性

问题：

可以同修改多个属性

问题：无法自定保护规则

解决:默认值三个都是false

要将writable: 改为true

enumerable: 改为true

configurable不用改

(2)访问器属性: 不直接存储数据

专门对其他属性提供保护的特殊属性

何时：只要需要自定义的保护规则保护一个属性时

如何：

内部属性: 不允许直接用.访问的属性

.\_\_proto\_\_ .class

正课:

1. \*\*\*\*\*ES5

1. 对象的属性: 对对象的属性提供了保护

命名属性:

数据属性:

四大特性:{

value: 值,

writable: true/false,

enumerable: true/false,

configurable: true/false

}

获取四大特性:

var attrs=

Object.getOwnPropertyDescriptor(obj,"属性名");

attrs:{ value: , writable: , ...}

修改四大特性:

Object.defineProperty(obj,"属性名",{

特性: 值,

configurable:false

})

如果对象的属性不存在，则自动添加。

特性的默认值:

直接量中的属性，特性默认值为true

defineProperty添加的属性，特性值都为false

同时修改多个属性的特性:

Object.defineProperties(obj,{

属性名1:{

四大特性:

},

...

})

问题: 无法自定义保护规则

解决:

访问器属性: 不直接保存属性值

专门保护其他属性的特殊属性

何时使用: 要用自定义规则保护一个属性时

如何:

四大特性: {

//专门负责读取受保护的属性值

get:function(){ return 属性值; },

//专门负责修改受保护属性值

set:function(value){

//判断value符合自定义规则的要求

属性名=value;

//否则

报错

},

enumerable: true/false,

configurable: true/false

}

添加访问器属性:

Object.defineProperty(obj,"属性名",{

get:function(){return xxx},

set:function(value){ xxx=value},

enumerable: true/false,

configurable: true/false

});

原理:

读取obj.访问器属性时，自动调用访问器属性的get方法

给obj.访问器属性赋值时，自动调用set方法，参数value自动获得要赋的新值。

问题:受访问器属性保护的值不能放在普通的数据属性中

解决: 使用闭包定义访问器属性和受保护的变量

内部属性: \_\_proto\_\_ class

extensible:true

2. 防篡改: 阻止对已经创建好的对象的属性结构进行篡改

保护整个对象

1. 防扩展: 禁止添加新属性。

判断obj是否可扩展: Object.isExtensible(obj)

阻止对obj的任何扩展: Object.preventExtensions(obj)

将对象的extensible: false

2. 密封: 在防扩展基础上，再禁止删除任何属性

判断obj是否密封: Object.isSealed(obj)

设置obj密封: Object.seal(obj)

将对象的每个属性configurable:false

何时: 强烈建议在构造函数结尾都要密封当前对象

3. 冻结: 在密封基础上，再禁止修改任何属性的值

判断obj是否被冻结: Object.isFrozen(obj)

冻结obj: Object.freeze(obj)

将对象的每个属性writable:false

何时: 如果一个对象的所有属性值都不允许修改时

3. Object.create(): 基于一个现有父对象

创建一个子对象

再扩展子对象的自有属性

何时: 如果先有父对象，再根据父对象创建子对象时

如何:

1. 仅创建子对象: var child=Object.create(father);

2. 创建子对象时，为子对象扩展自有属性:

　　　 var child=Object.create(father,{

属性名:{

四大特性

},

...

})

4. 数组API:

判断: 数组中的元素是否满足指定的条件

arr.every(): 检查数组中每个元素是否都满足条件

arr.some(): 检查数组中是否包含满足条件的元素

如何:

var bool=arr.every(function(val,idx,arr){return 判断结论})

every会拿function到每个元素上执行一次，获得每个元素的判断结果。

——所有判断结果都为true，最后才返回true

val: 自动获得当前元素值

idx: 自动获得当前位置

arr: 指当前正在遍历的数组

遍历: 对每个元素执行相同的操作

arr.forEach(): 对原数组中每个元素执行相同的操作

var newArr=arr.map():

取出原数组中每个元素的值，执行相同的操作后，放入一个新数组中返回

如何:

arr.forEach(function(val,idx,arr){

arr[idx]=新值;

})

Day09

正课:

1. \*\*\*\*\*ES5:

数组API: 过滤和汇总

bind()

严格模式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. 数组API:

过滤: 选取出原数组中符合条件的元素，组成新数组返回

var subArr=arr.filter(function(val,i,arr){

return 判断条件

})

var arr=[1,2,3,4,5];

var evens=arr.filter(function(val){

return val%2==0;

});

console.log(String(arr));

console.log(String(evens));

汇总: 依次统计数组中每个元素的值，最终得出一个结果

var result=arr.reduce(function(prev,val,i,arr){

//prev: 获得的是截至目前的临时汇总至

return prev和val的汇总值。

}, startVal)

var sum=arr.reduce(function(prev,val){

return prev+val;

});

console.log(sum);

var arr2=[2,4,6,8,10];

sum=arr2.reduce(function(prev,val){

return prev+val;

},sum);

console.log(sum);

2. bind()

什么是: 基于一个现有函数，创建一个新函数，并提前绑定this

为什么: call和apply，都是临时借用，不是永久绑定。

何时: 只要希望一个函数中的this，始终固定指向一个对象。

如何:

function 原函数(参数1,参数2,...){...}

var 新函数=

原函数.bind(替换this的对象[, 提前绑定的部分参数])

新函数(剩余参数值) //新函数不用再用call/apply绑定this

if(Function.prototype.bind===undefined)

Function.prototype.bind=function(obj/\*,args1\*/){

//arguments

var fun=this;//this->calc fun->calc

var args1=//保存除obj外的剩余参数到args1

Array.prototype.slice.call(arguments,1);

//相当于: arguments.slice(1)

return function(/\*args2\*/){

//arguments

var args2=//复制后传入的新参数到args2

Array.prototype.slice.call(arguments);

//相当于arguments.slice();//复制所有元素

//将新旧参数拼起来，一起传给fun

fun/\*calc\*/.apply(obj,args1.concat(args2))

}

鄙视:

区分call apply bind

call和apply: 临时借用函数，并替换函数中的this

调用函数

参数： call: 所有参数单独传入

apply: 所有参数放入数组中集中传入

bind: 创建函数, 永久绑定函数中的this

创建函数

3. 严格模式: 比普通js运行模式更严格的运行模式

新项目最好都在严格模式下开发

旧项目应该逐个方法向严格模式迁移

如何: 在代码段开头: "use strict";

加在script的开头，整个script启用严格模式——新项目

加在单个function的内部开头，仅对当前函数启用严格模式

——旧项目

要求:

1. 将静默失败升级为了错误！

2. 禁止对未声明的变量赋值！

3. 增加了eval作用域

4. 不建议使用arguments.callee或caller

Ex:

var arr=[1,2,3,4,5];

var evens=arr.filter(function(val){

return val%2==0;

});

console.log(String(arr));

console.log(String(evens));

var sum=arr.reduce(function(prev,val){

return prev+val;

});

console.log(sum);

var arr2=[2,4,6,8,10];

sum=arr2.reduce(function(prev,val){

return prev+val;

},sum);

console.log(sum);

ex2:

var fun=this;//先将原函数calc，另存为到变量fun

//bind方法返回一个新的内层函数对象

return function(){

//arguments

//在新内层函数对象中借用fun引用的原函数

//用指定对象obj替换fun引用的原函数中的this

//接收所有传入新函数的参数，打散后传给fun函数

fun.apply(obj,arguments);//obj.calc();

}//闭包中永久固定保存了两个局部变量:fun和obj