正课:

补: \*\*\*关联数组

1. 数组API:

2. 栈和队列:

3. 二维数组:

什么是索引数组: 下标都为数字的数组

\*\*\*关联数组:

什么是: 可自定义下标名称的数组

为什么: 为了让每个元素都有一个专门的名称:

好处: 1. 便于维护

2. 便于查找

何时: 1. 希望每个元素都有专门的意义时

2. 希望快速查找元素时

如何: 2件事：

创建: 2步:

1. 先创建空数组: var ym=[];

2. 向空数组中添加新元素，要使用自定义的下标名称

ym["name"]="杨幂"

ym["math"]=81;

ym["chs"]=59;

ym["eng"]=89;

访问: 访问关联数组和访问索引数组完全一样

只不过要使用自定义的下标名称

关联数组中length属性失效，永远等于0

因为关联数组中没有数字下标,length无法+1

遍历关联数组:

for(var key in ym){//in依次取出ym中每个可以

key //当前房间号

ym[key]//当前元素值

强调: key不能加引号, 因为每次循环都在变化

}

关联数组原理:

问题: 索引数组，无法提前预知元素的具体位置

只能遍历查找: 受元素个数和元素存储位置影响

解决: hash算法: 根据一个字符串，计算出尽量不重复的一个序号。相同的字符串计算出的序号一定相同

向关联数组中保存元素:

将自定义下标名交给hash算法，计算一个散列的位置序号。将元素保存到序号指定位置

从关联数组中取值时:

将自定义下标名交给hash算法，计算出和存入时完全相同的序号，引擎直接去序号位置获得元素。

优: 不受存储位置和元素个数的影响

1. 数组API:

什么是数组: 连续保存一组数据，并提供了操作数据的简化API

API: 别人已经实现的，咱们用现成的程序

包括:

转字符串: 2种:

String(arr): 将arr中每个元素都转为字符串，用逗号连接

何时: 给数组拍照, 用于比较操作前后数组是否发生了变化。

arr.join("连接符"):将arr中每个元素都转为字符串，用自定义的连接符连接元素

何时: 只要希望使用逗号以外的其它连接符时

固定套路: 1. 把单词拼接为句子:

2. 无缝拼接:

错误: arr.join() => String(arr) =>逗号连接

正确: arr.join("")

重要用途: 判断数组是否为空！

3. 动态生成页面元素

console.log(String(arr)),console.log(arr),console.dir(arr)

console.log(arr); 先输出dir的结构，刷新后变为String(arr)

仅输出内容，不关心结构: console.log(String(arr))

查看数组存储结构: console.dir(arr)

dir查看结果时，不遵循先后执行的顺序

拼接和选取: 都无权修改原数组，只能返回新数组对象

拼接: 将其他数据或另一个数组，和当前数组拼接为新数组。

如何: var newArr=arr1.concat(值1,值2,arr2,...);

强调: 1. 不修改原数组，只返回新数组

2. 可打散数组类型参数为单个数据,再拼接

——concat独有！

选取: 复制原数组中指定开始位置到结束位置之间的元素，组成新数组——原数组保持不变

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

强调: 1. 如果一个函数两个参数都是下标时，就会含头不含尾。

简写: 1. 如果位置离结尾近，可用负数下标:

arr.length-n 可简写为: -n

2. 可省略第二个参数，表示一直选取到结尾

3. 两个参数都可省略: 完整复制一个数组

修改数组: splice

强调: 直接修改原数组

删除元素: arr.splice(starti,n)

删除starti位置开始的n个元素

强调: 不必考虑含头不含尾

其实: 有返回值: 返回被删除的元素组成的临时数组

var deletes=arr.splice(starti,n);

简写: 1. 支持负数下标，表示倒数第n个

2. 省略n

插入新元素: arr.splice(starti,0,值1,值2,...)

在starti位置插入值1,值2,...

原starti位置的值向后顺移

强调: 不支持打散数组参数

替换: 先删除旧的，再在同一位置插入新的

arr.splice(starti,n,值1,值2,...)

先删除starti开始的n个元素

再在starti位置插入值1,值2,...

强调: 删除的个数和插入的个数不必相同

翻转: arr.reverse();

\*\*\*排序:

什么是: 将数组中的元素按从小到大或从大到小的顺序重新排列

何时: 所有要给用户展示的数据必须都要先排序

如何: arr.sort();

默认排序规则: 将所有元素都转为字符串，再按字符串升序排列。

何时: 只有按字符串升序排列时，才用默认的sort()

问题: 只能按字符串升序排列

解决: 自定义比较器函数:

什么是: 专门比较任意两值大小的函数

要求: 2个参数: a,b

返回值: 如果a>b，就返回正数

如果a<b，就返回负数

否则返回0

最简单的数字升序比较器:

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

如何使用: 将比较器函数对象作为参数传入sort()

arr.sort(cmp);

强调: 不加()

回调: 自己定义的函数，自己不调用

而是传入另一个函数中，被另一个函数反复调用。

何时: 只要对数字元素排序，都要自定义比较器函数

问题2: 如何降序:

解决: 只要颠倒比较器结果的正负号，就可改升序为降序

最简单的数字降序比较器:

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

2. 栈和队列:

说明: js中没有专门的栈和队列结构，都是用普通数组模拟的

栈stack:

什么是栈: 一端封闭，只能从另一端进出的数组

何时: 只要希望始终使用最后进入数组的新元素时

如何:

1. 结尾出入栈:

结尾入栈: arr.push(值) => arr[arr.length]=值

强调: 1. 其实push可压入多个值

2. 不支持打散数组参数

结尾出栈: var last=arr.pop()

2. 开头出入栈:

开头入栈: arr.unshift(值)

强调: 开头入栈后的元素顺序和结尾入栈后的元素顺序是相反的。

开头出栈: var first=arr.shift();

总结: 向数组中添加元素4种:

1. concat(): 1. 不修改原数组，返回新数组

2. 在结尾拼接元素

3. 支持打散数组类型参数

2. splice(): 1. 直接修改原数组

2. 在任意位置插入新元素

3. 不支持打散数组类型参数

3. push(): 1. 直接修改原数组

2. 只能在结尾拼接元素

3. 不支持打散数组类型参数

4. shift(): 1. 直接修改原数组

2. 只能在开头拼接元素

3. 不支持打散数组类型参数

取出数组元素: 4种:

1. slice(): 1. 可获取任意位置的任意个元素

2. 不修改原数组，返回选中的元素组成的新数组

2. splice(): 1. 删除任意位置的任意个元素

2. 直接修改原数组

3. 返回被删除的元素组成的新数组

3. pop(): 1. 只能从结尾删除一个元素，并返回

4. shift(): 1. 只能从开头删除一个元素，并返回

队列queue:

什么是: 只能从结尾进入，从开头出的数组

何时: 只要希望按照先来后到的顺序使用数组元素时

如何:

1. 从结尾入队列: arr.push(值)

2. 从开头出队列: 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: 2种:

1. 保存横行竖列的二维数据

2. 一个大的数组中，还需要对元素进行更细致分类

如何:

创建: 2种:

1. 先创建空数组，再添加子数组:

var arr=[];

arr[0]=[0,0,0,0];

arr[1]=[0,0,0,0];

2. 创建数组同时，初始化子数组

var arr=[

[0,0,0,0],

... ,

[0,0,0,0]

];

访问: arr[r][c] 每个元素的用法和普通数组的元素完全一样

越界: 二维数组的行下标r，不能越界

遍历二维数组: 外层循环控制行，内层循环控制列

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

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

data[r][c] //当前正在变量的元素

}

}