正课:

1. \*\*\*关联数组:

2. \*\*数组API

3. \*\*\*排序

4. 栈和队列

5. 二维数组

1. \*\*\*关联数组:

什么是: 可自定义下标名称的数组

vs 索引数组: 下标都为默认的数字的数组

稀疏数组: 下标不连续的数组

为什么: 索引数组的数字下标，没有意义，不便于快速查找和维护

何时: 希望通过下标名称，快速获得想要的数组元素时

如何:

创建: 2步:

1. 先创建空数组

2. 向数组中添加新元素，要使用自定义的下标名称:

强调: length=0，失效

访问元素: 数组名["自定义下标名称"]

数组中的每个元素用法和普通变量完全一样！

优势: 极快的查找数组的一个元素

和数组中元素的个数，以及元素存储位置无关！

vs 索引数组: 以遍历的方式查找数组元素

缺点: 1. 受数组元素个数影响. 2. 受元素存储位置的影响

遍历: for(var key in hash){//in会依次取出hash中每个key

key //下标名称

hash[key] //当前正在遍历的元素内容

}

强调: hash[key] 不要加""， 因为key是个变量，每循环依次都会变化

2. \*\*数组API:

什么是API: 别人已经定义好的，咱们直接用现成的的程序

为什么: 简化开发，复用功能！

何时: 今后，使用任何一种类型时: Array, String, Date... 都要学习每种类型提供的简化版API。

包括:

1. 转字符串: 2种:

1. var str=String(arr); 将arr的每个元素都转为字符串，用逗号相连.——拍照

何时: 今后，只关心数组当前的内容时

2. var str=arr.join("自定义的连接符")

将arr的每个元素都转为字符串, 用自定义的连接符连接每个元素.

何时: 今后如果要求转换字符串之后的格式时

固定套路:

1. 将单词数组拼接为句子: words.join(" ")

2. 无缝拼接: 将字母拼成单词:

错误: chars.join() => String(chars)

正确: chars.join("")

3. 如何判断一个数组是空数组:

无缝拼接后，依然是空字符串的，说明是空数组

arr.join("")===""

**4. 动态生成页面元素的内容**

**id.innerHTML=**

**"<ANY>"+arr.join("</ANY><ANY>")+"</ANY>"**

2. 拼接和选取:

强调: 无论是拼接，还是选取操作，都不能修改原数组，而是返回新数组

1. 拼接: 将其他数组或元素，拼接到当前数组的末尾，组成新数组

var arr3=arr1.concat(值1,值2,...,arr2,...)

当拼接另一个子数组时，concat会大散数组参数为单个元素，再逐个拼接。

2. 选取: 复制出原数组中指定位置的几个元素值，组成新数组.

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

强调: 凡是两个参数都是下标位置的函数，都含头不含尾.

简写:

1. 可用负数参数直接代表倒数第n个位置:

arr.slice(starti,-n+1); 截取到倒数第n个位置

原理: arr.slice(starti,arr.length-n+1);

2. 省略第二个参数，表示到结尾！

3. 两个参数都省: 表示从头到尾完整复制整个数组

3. 修改数组: 插入，删除，替换

强调: 直接修改原数组

1. 删除: arr.splice(starti,n); 删除arr中starti开始的n个元素

强调: 不考虑含头不含尾,因为第二个参数不是下标

简写: 1. 省略第二个参数: 一直删到结尾

2. 支持负数参数: 表示倒数第n个位置

其实splice有返回值: 返回被删除元素组成的临时新数组

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

强调: 即使仅删除一个元素，也会放在数组中返回。

2. 插入: arr.splice(starti,0,值1,值2,...)

原插入位置及其之后的元素，依次向后顺移

vs concat: 1. splice可在任意位置插入新元素

concat只能在末尾拼接

2. splice直接修改原数组

concat不修改原数组，返回新数组

3. splice不支持大散数组参数

concat 支持大散数组参数为单个元素

3. 替换: 先删除元素，再在原位置插入新元素

arr.splice(starti,n,值1,值2,...); 先删除n个，再插入新值

强调: 删除的元素个数和插入的新元素个数不必相同！

4. 翻转: arr.reverse();

3. \*\*\*排序:

什么是: 将数组中的元素，按从小到到或从大到小的顺序重新排列

为什么: 便于快速查找和维护

何时: 今后只要多个数据显示给用户前，必须都要先排序!

如何:

1. 排序算法: 冒泡，插入，快速

冒泡: (默认升序排列)

基本: 从0开始遍历数组中每个元素，不用到头

只要当前元素>下一个元素

就交换两数位置

问题: 一轮遍历，无法完成排序

解决: 需要用外层循环控制多轮遍历

2. 现成的API: arr.sort();

问题: arr.sort只能对字符串或一位数字排序

无法对多位数字类型排序

原因: arr.sort()默认先将每个元素转为字符串，再排序

解决: 自定义比较器函数:

什么是比较器函数: 专门比较任意两数大小的函数

何时: 只要用arr.sort() 对数字类型的元素进行排序时，都要自定义比较器函数

如何: 2步:

1. 定义比较器函数:

function compare(a,b){

return a-b;

}

2. 将比较器函数作为参数，传递给arr.sort()，让arr.sort()用于比较每个元素的值.

arr.sort(compare);

强调: 不要加()

回调: 将一个函数交给另一个函数去自动调用

不加(): 1. 不止调用一次！

2. 每次调用，传什么参数，不确定！

3. 升序改降序:

只要颠倒比较器函数返回值的正负号

4. 栈和队列:

js中没有专门的栈和队列类型，都是用数组+特殊API模拟的

栈(stack):

什么是: 一端封闭，只能从另一端进出的数组

特点: FILO

何时: 只要希望按照进入数组的反向顺序，使用数组元素时

如何:

1. 开头封闭, 结尾出入栈:

入栈: arr.push(值) 简化了arr[arr.length]=值

出栈: var last=arr.pop();

2. 结尾封闭，开头出入栈:

入栈: arr.unshift(值)

强调: 开头入栈的顺序和结尾入栈的结果相反！

何时: 只要希望获得与结尾入栈相反的元素顺序时

出栈: var first=arr.shift();

队列(queue):

什么是: 只能从结尾进入，从开头出的数组

特点: FIFO

何时: 只要希望按先来后到的顺序使用数组元素时

如何:

结尾入队列: arr.push(值)

开头出队列: var first=arr.shift();