数据结构和算法

数据结构(静态的，一个结果，存数据的)和算法(动态的，处理事务过程，操作数据结构的)的概念:

数据结构:

可以容纳数据的结构被称为数据结构，比如:

现实中我们如果要搬家，那么一些衣服、家具之类的肯定要找个容器装起来，那么衣服和家具就是数据，而这个容器被称为数据结构。

算法：

算法是用来对数据结构处理的方法，比如:

衣服、家具在搬运的过程中可能需要横着搬，或者竖着搬，这种搬运的过程被称为算法，处理问题的方法就是算法。

数据结构和算法的关系就是：

数据结构就是一堆等待处理的数据像现实中的衣服、家具，我们把他打包起来等待着被处理

算法就是处理数据结构的，比如拧魔方（数据结构，静态的等待着被处理），拧魔方的过程、方法（动态的，算法）

线性数据结构-数组(最基本的数据结构)

其实在前端中数组是定长的，只不过你达到一定的数量它自动给你扩容了，这是JS的引擎帮你实现的，但是不能不管，扩容的过程是消耗性能(CPU)的，好的程序员它能在程序运行的过程中，能提前设定好数组的长度，不会产生性能的消耗。

1. 扩容其实就是创建一个新的空间，把之前的数据复制过来，那么为什么不直接在后面扩容非要从新创建呢？因为内存空间是操作系统分配的，它不一定分配在哪一块，只能是声明我想要一块连续的多大的空间，你说我如果声明只想要一块空间，操作系统指不定给你把扩容的空间随便找个地方去放，比如放在你原数组的中间，那么第九个数据就存不进去了，因为数据讲究的是顺序，所以说声明申请空间是操作系统做的，所以说当你的数组存不下的时候JS就会向操作系统申请我想要一块连续的16的空间。

(2) 扩容的实现:比如一个数组长度为8，那么我如果要往里面塞第九个值，JS会先向操作系统申请我想要一块连续的长度为16(长度为多少不固定，可能会是16操作系统会设置提前量)的空间，操作系统会从新开辟一个空间，然后把刚才的8个给复制过来，最后才把第九个数存进去，这是扩容的过程。

(3) 所以说在使用数组的时候先预估下自己的数组有多长，因为自己的业务这是能够办到的，比如：贪吃蛇，我大概算一下大部分人到100个就挂掉，那么我可以先声明出120长度的，我让玩家在不到120的时候就挂掉，那么就可以保证我程序的完美运行，不至于发生扩容现象。

一维是线，二维是面，三位是立体，四位是立体带空间

所以说线性数据结构也被称为一维数据结构

1. 一维数据结构（线性数据结构）
   1. 把一堆数据给存起来，类似饼干盒，把所有的饼干放进盒子里。
   2. 线性数据结构强调的是存储与顺序。
   3. 它没有什么特性，最广泛的应用就是存数据。
   4. 特性:
      1. 数组定长，存储在物理空间上是连续的，
      2. 数组定长它刚开始设定多少就是多少，底层数组的长度是不可变的。
      3. 比如:a={1,2,3,4,5,6,7,8}，问数组的变量指向哪里？指向了数组第一个元素的位置,指向数组的起点而不是整个数组，a[1],a[2],a[3] 方括号表示存储地址的偏移，操作系统小知识：通过偏移查询数据性能最好。
      4. 数组的优点一：查询性能好，指定查询某个位置。
      5. 数组的缺点一：
         1. 因为空间必须得是连续的，所以如果数组比较大，容易存不下。(比如我家有个衣柜特别的大而且还不能拆，你说要是能拆还能拆下来找个地方堆起来。)当系统的空间碎片较多的时候，容易存不下。
         2. 空间碎片：我们电脑里的每一个程序都要声明变量，变量就会占用系统的存储空间，当这个变量在销毁的时候，这块空间会滞空，系统在分配的空间的时候肯定是挨排分配的，第一个位置是xxx，第二个位置是xxx,但是用着用着第二个位置可能不用了，是空着的是断开的，比如我一个数组有10个位置，但是中间3 和 6 断开了，按理说我们一个长度为3个数据存到数组里是可以存的下的，但是因为数组中间断开了，数组的存储必须是连续性的，我们把这些小的断开的空闲的空间称为空间碎片，所以说数组在遇到空间碎片的时候就很尴尬，明明空间感觉够用，但是我却放不下一个数组，这也是大数组的缺点，所以说我们在编程的时候要避免数组过大，不要太长，因为如果过长万一哪个数不用了中间断开了，形成大量的空间碎片了，那么系统为了让你能够继续存的下，他就开始整理空间，一整理空间就非常耗费CPU，非常的缓慢 你性能，能好吗？
         3. 数组的缺点二：因为数组的长度是固定的，所以数组的内容难以被添加和删除。为什么？比如1,2,3,4,5,6,7,8 我把 5 删了1,2,3,4, ,6,7,8 那么你下标为[4]的地方就空了，但是不能为空所以系统会把它往前面移一位1,2,3,4,5,6,7, 把末尾的删了，所以说 代价太大了，太耗费性能了。

如何正确在JS中创建一个数组

