Java数据结构与算法

1. **数据结构**
2. **线性结构**

线性结构作为最常用的数据结构，其特点是数据元素之间存在一对一的线性关系。

线性结构有两种不同的存储结构，即顺序存储结构和链式存储结构。顺序存储的线性表称为顺序表，顺序表中的存储元素是连续的

链式存储的线性表称为链表，链表中的存储元素不一定是连续的，元素节点中存放数据元素以及相邻元素的地址信息

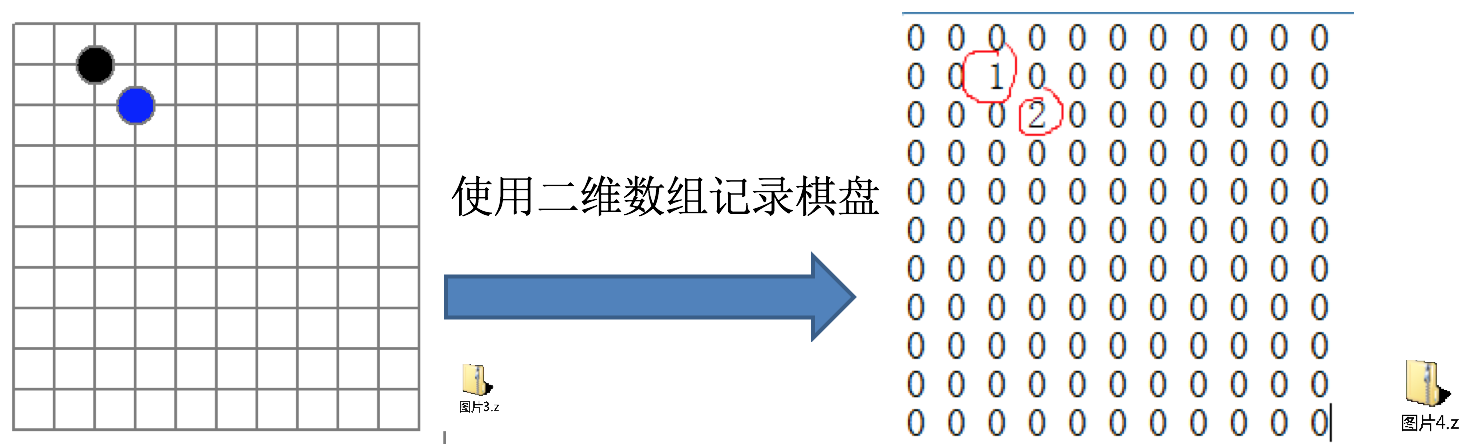
线性结构常见的有：数组、队列、链表和栈

1. **非线性结构**

非线性结构包括：二维数组，多维数组，广义表，树结构，图结构

1. **数据结构和算法**
2. **稀疏数组**

编写的五子棋程序中，有存盘退出和续上盘的功能



**分析问题**：因为该二维数组的很多值是默认值0, 因此记录了很多没有意义的数据.->稀疏数组。

当一个数组中大部分元素为０，或者为同一个值的数组时，可以使用稀疏数组来保存该数组。

稀疏数组的处理方法是:

1）、记录数组一共有几行几列，有多少个不同的值。

2）、把具有不同值的元素的行列及值记录在一个小规模的数组中，从而缩小程序的规模。

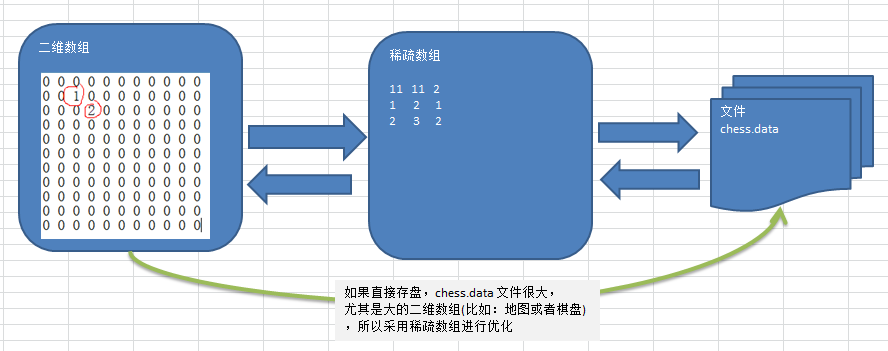


**应用实例**：

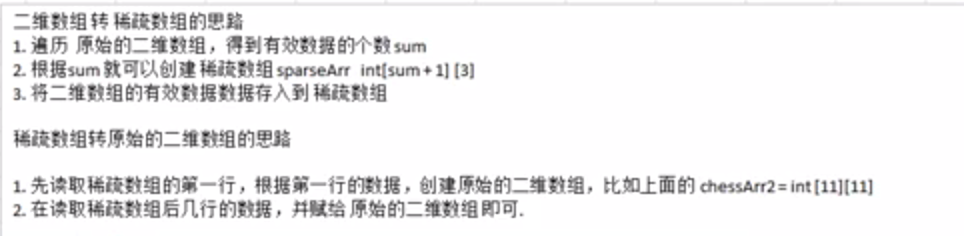
1）、使用稀疏数组，来保留类似前面的二维数组(棋盘、地图等等)

2）、把稀疏数组存盘，并且可以从新恢复原来的二维数组数

3）、整体思路分析



实现思路：



1. **队列**

队列类似于去银行办理业务排队的场景

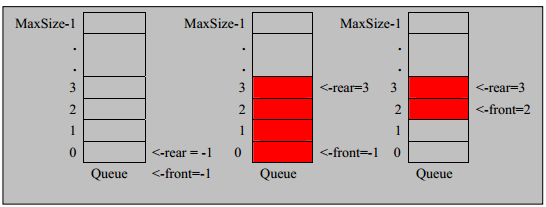
队列是一个有序列表，可以用数组或是链表来实现。

遵循先入先出的原则。即：先存入队列的数据，要先取出。后存入的要后取出

**数组实现**

队列本身是有序列表，若使用数组的结构来存储队列的数据，则队列数组的声明如下图, 其中 maxSize 是该队列的最大容量。

因为队列的输出、输入是分别从前后端来处理，因此需要两个变量 front及 rear分别记录队列前后端的下标，front 会随着数据输出而改变，而 rear则是随着数据输入而改变，如图所示:



注：

rear 是队列最后[含]

front 是队列最前元素[不含]

**思路**：

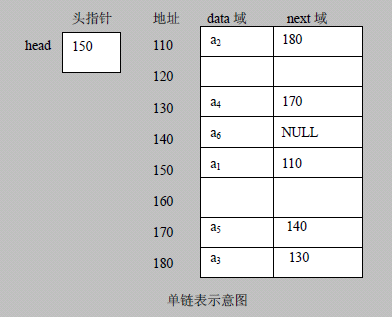
当我们将数据存入队列时称为”addQueue”，addQueue 的处理需要有两个步骤：思路分析

1）、将尾指针往后移：rear+1 , 当front == rear 【空】

2）、若尾指针 rear 小于队列的最大下标 maxSize-1，则将数据存入 rear所指的数组元素中，否则无法存入数据。 rear == maxSize - 1[队列满]

1. **链表**

链表是有序的列表，但是它在内存中是存储如下



1)、链表是以节点的方式来存储,是链式存储

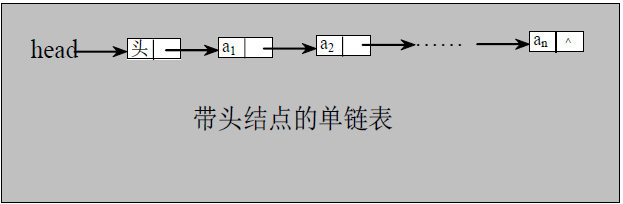
2）、每个节点包含 data 域， next 域：指向下一个节点.

3）、如图：发现链表的各个节点不一定是连续存储.

4）、链表分带头节点的链表和没有头节点的链表，根据实际的需求来确定

**单链表**

(带头结点) 逻辑结构示意图如下



单链表逆序打印：

最好的方法：可以利用栈这个数据结构，将各个节点压入到栈中，然后利用栈的先进后出的特点，就实现了逆序打印的效果。涉及的类：java.util.Stack

**双向链表**

单向链表的缺点分析:

1）、单向链表，查找的方向只能是一个方向，而双向链表可以向前或者向后查找。

2）、单向链表不能自我删除，需要靠辅助节点（也就是删除时，总是找到temp，temp是待删除节点的前一个节点） ，而双向链表，则可以自我删除。

示意图帮助理解删除

双向链表的遍历、添加、修改、删除的操作思路

1）、遍历和单链表一样，只是可以向前，也可以向后查找

2）、添加（默认添加到双向链表的最后）

先找到双向链表的最后这个节点temp

temp.next = newNode

newNode.pre = temp

3)、修改思路、原理跟单向链表一样

4）、删除

因为是双向链表，因此，我们可以实现自我删除某个节点

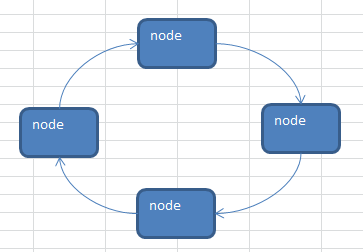
直接找到要删除的这个节点，比如temp

temp.pre.next = temp.next

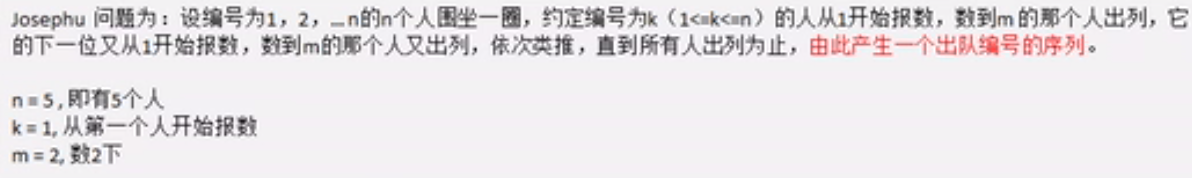
temp.next.pre = temp.pre

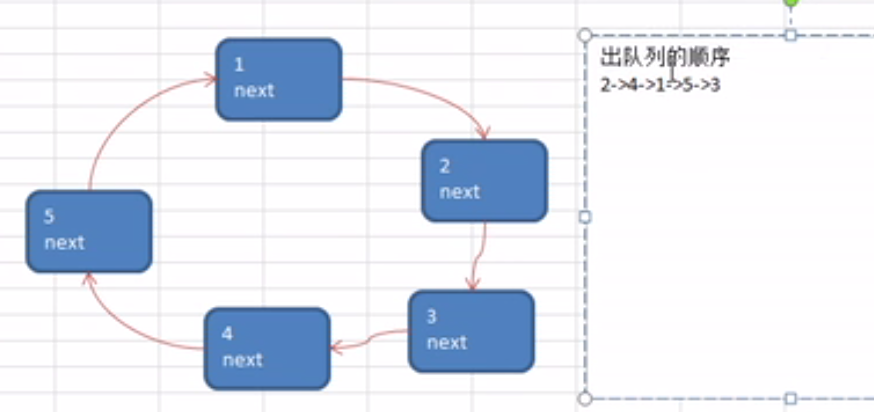
**单向环形链表**

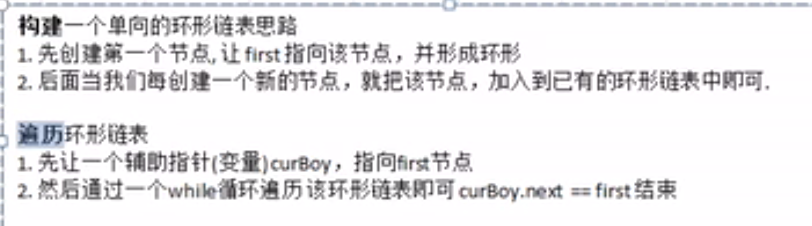
示意图（不带头结点）



**约瑟夫问题**：







构建一个单向的环形链表思路

1）、先创建第一个节点，让first指向该节点，并形成环形

2）、后面当我们每创建一个新节点，就把该节点，加入到已有的环形链表中即可。

遍历环形链表：

1）、先让一个辅助指针curNode，指向first节点

2）、然后通过一个while循环遍历该环形链表即可，当curNode.next == first结束遍历

1. **栈**

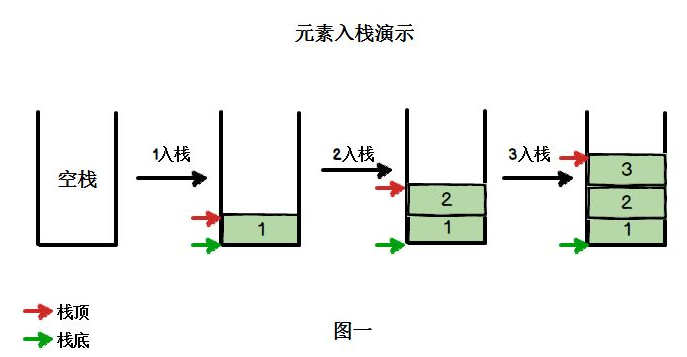
栈的英文为(stack)

栈是一个先入后出(FILO-First In Last Out)的有序列表。

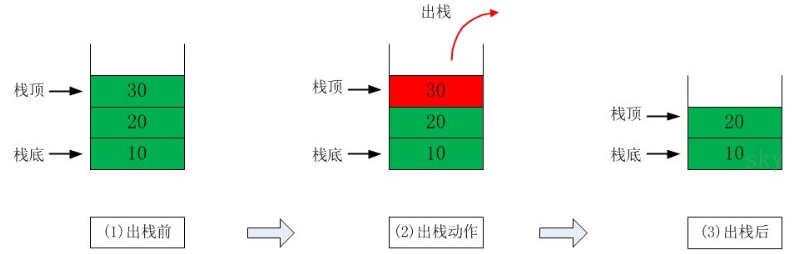
栈(stack)是限制线性表中元素的插入和删除只能在线性表的同一端进行的一种特殊线性表。允许插入和删除的一端，为变化的一端，称为栈顶(Top)，另一端为固定的一端，称为栈底(Bottom)。

根据栈的定义可知，最先放入栈中元素在栈底，最后放入的元素在栈顶，而删除元素刚好相反，最后放入的元素最先删除，最先放入的元素最后删除。

入栈push



出栈pop



应用场景

1）、子程序的调用：在跳往子程序前，会先将下个指令的地址存到堆栈中，直到子程序执行完后再将地址取出，以回到原来的程序中。

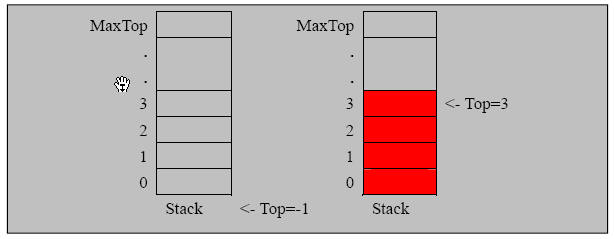
2）、处理递归调用：和子程序的调用类似，只是除了储存下一个指令的地址外，也将参数、区域变量等数据存入堆栈中。

3）、表达式的转换[中缀表达式转后缀表达式]与求值(实际解决)。

4）、二叉树的遍历。

5）、图形的深度优先(depth一first)搜索法。

**使用数组模拟栈**



实现栈的思路分析：

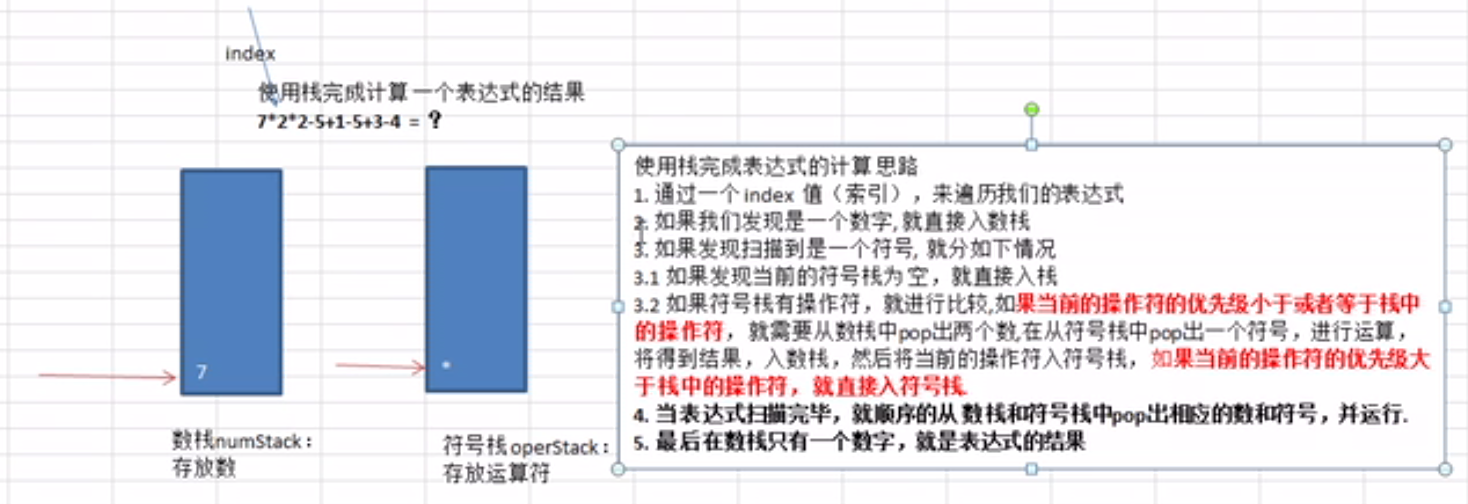
1）、使用数组来模拟栈

2）、定义一个top来表示栈顶，初始化为-1

3）、入栈的操作，当有数据加入到栈时，top++;stack[top] = data;

4）、出栈的操作，int value = stack[top];top--;return value;

**栈实现表达式计算**



**表达式计算**

前缀表达式又称波兰式，前缀表达式的运算符位于操作数之前

**举例说明**： (3+4)×5-6 对应的前缀表达式就是 - × + 3 4 5 6

从右至左扫描表达式，遇到数字时，将数字压入堆栈，遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算（栈顶元素 和 次顶元素），并将结果入栈；重复上述过程直到表达式最左端，最后运算得出的值即为表达式的结果

例如: (3+4)×5-6 对应的前缀表达式就是 - × + 3 4 5 6 , 针对前缀表达式求值步骤如下:

1）、从右至左扫描，将6、5、4、3压入堆栈

2）、遇到+运算符，因此弹出3和4（3为栈顶元素，4为次顶元素），计算出3+4的值，得7，再将7入栈

3）、接下来是×运算符，因此弹出7和5，计算出7×5=35，将35入栈

4）、最后是-运算符，计算出35-6的值，即29，由此得出最终结果

**中缀表达式**

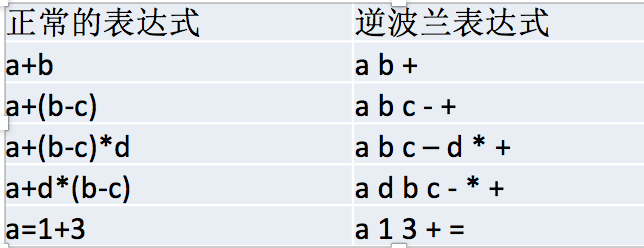
中缀表达式就是常见的运算表达式，如(3+4)×5-6

中缀表达式的求值是我们人最熟悉的，但是对计算机来说却不好操作(前面我们讲的案例就能看的这个问题)，因此，在计算结果时，往往会将中缀表达式转成其它表达式来操作(一般转成后缀表达式.)

**后缀表达式**

后缀表达式又称逆波兰表达式,与前缀表达式相似，只是运算符位于操作数之后

举例说明： (3+4)×5-6 对应的后缀表达式就是 3 4 + 5 × 6 –



从左至右扫描表达式，遇到数字时，将数字压入堆栈，遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算（次顶元素 和 栈顶元素），并将结果入栈；重复上述过程直到表达式最右端，最后运算得出的值即为表达式的结果

例如: (3+4)×5-6 对应的后缀表达式就是 3 4 + 5 × 6 - , 针对后缀表达式求值步骤如下:

1）、从左至右扫描，将3和4压入堆栈；

2）、遇到+运算符，因此弹出4和3（4为栈顶元素，3为次顶元素），计算出3+4的值，得7，再将7入栈；

3）、将5入栈；

4）、接下来是×运算符，因此弹出5和7，计算出7×5=35，将35入栈；

5）、将6入栈；

6）、最后是-运算符，计算出35-6的值，即29，由此得出最终结果

1. **Aa**
2. **Aa**