# 道纲

有物混成，先天地生，寂兮廖兮，独立而不改，周行而不殆，可以为天地母。吾不知其名，强字之曰道，强为之名曰大。大曰逝，逝曰远，远曰反。故道大，天大，地大，人亦大。域中有四大，而人居其一焉。人法地，地法天，天法道，道法自然！

世间万物皆不断变化，技术更是不断更新迭代，而人若不背道而驰，则必须不断学习，此为顺势而为，合道。

而学习的本身除了学会如何使用某种技术，更重要的是如何去学习新东西，如何持续学习，如何超前了解并掌握未来前景。

人世间最大的悲哀不就是浪费了年轻的大好时光吗？

# 算法和数据结构

简介：

算法是程序的灵魂，优秀程序可以在处理海量的同时，依旧可以保持高速运算！

数据结构是研究组织数据方式的学科！数据结构是作为算法的基础！

## 数据结构

数据结构的定义：逻辑上数据的组织方式和处理方式，数据的物理存储方式很明显就是一个地址空间一个地址空间的存放。而逻辑上的数据组织方式有很多，树，图，队列，哈希等！

哈希是根据某种算法能够快速定位数据的组织方式！

数据的处理方式：数据的增删改查以及遍历

数据结构包括线性结构和非线性结构

* 线性结构：最常用的数据结构，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方式，即顺序存储和链式存储，顺序存储称之为顺序表，顺序表中的元素是连续的
  + 链式存储的线性表称之为链表，链表的逻辑结构上是连续的，实际物理存储地址上是不一定连续的！

常见的线性结构：数组，队列，链表，栈！

非线性数据结构：多维数组，广义表，树结构，图结构

* 树：0至一个前驱和0至n个后继的一种数据结构，树的结构比较稳定均衡，在计算机领域得到广泛应用。
* 图：0至n个直接前驱和直接后继，
* 哈希结构：没有直接的前驱和后继，哈希结构通过某种特定的哈希函数将索引和纯纯的值关联起来，是一种查找效率非常高的数据结构

如何衡量数据处理性能：数据结构的复杂程度分为空间复杂度和时间复杂度，在存储设备越来越便宜的情况下，时间复杂度成为重点考量的因素。

例如O（n^3）表示程序执行时间随着输入规模程序3次方倍增长，这就是比较差的算法实现。

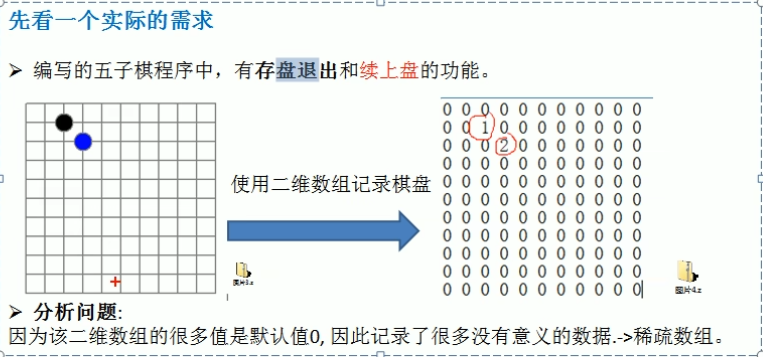
时间复杂度由好到差排序：

* O(1)>O(logn)>O(n)>O(nlogn)>O(n^2)>O(n^3)>O(2^n)

数据结构的各种类型有其优势也必然有其劣势，根据不同的场景和不同的需求他们也会有着不同的应用。

### 稀疏数组

实际需求：五子棋程序



稀疏数组：当一个数组中大部分元素都是0或者为同一个值时，可以使用稀疏数组来保存该数组。

稀疏数组的处理方法：

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以及值记录在一个小规模数组中，从而缩小程序规模



稀疏数组0行存储数据表示：有6行7列，8个数据

其他行记录值以及值的行列坐标！

二维数组转化稀疏数组思路：

* 遍历原始的二维数组，得到有效数据的个数sum
* 根据sum的值创建稀疏数组(sum+1)\*3
* 将有效数据存入稀疏数组

稀疏数组转原始数组思路

* 先读取稀疏数组的第一行，根据第一行数据创建原始二维数组
* 再读取后几行的数据并赋值给原始的二维数组即可！

代码演示：D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\sparsearr\SparseArr.java

获得稀疏数组：

两次遍历，第一次遍历是为了得到数组的有效数从而确定稀疏数组的行数，

第二次遍历是为了得到每个有效数据的行列值，代码中sum重置是作为信标复用！

获得原始数组：

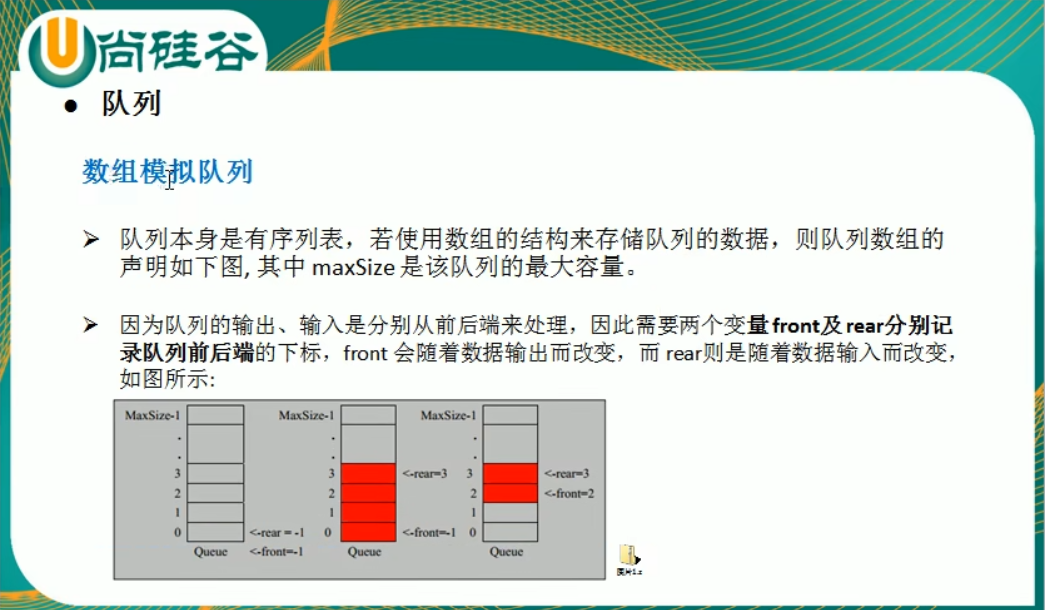
先读取第一行创建原始数组，再遍历有效数据进行赋值！

### 队列

一种顺序数据结构，特点：先入先出！实现方式：数组或者链表！

数组实现队列示例：D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\queue

实现分析：队列的输入和输出对应着队列的头部和尾部，因此需要连个指针分别指向队列的头和队列的尾，分别是front和rear。数据增加rear改变，取出数据front改变！



环形队列

思路：环形队列，假设环有10个空间，那么头尾指针移动一位时应该是(rear+1)%10!

头尾指针默认都处于0位置！

存放一位数据时，头指针还是0，尾指针挪一位为1！

存放两位数据时，头指针还是0，尾指针指向2！

………尾指针=（rear+1）%10

存放9位数据，头指针还是0，尾指针指向9

存放10位数据，头指针还是0，尾指针为0

为满：（尾+1）%10 = 头

为空：头 == 尾

损耗一个空间用于存放尾指针！

### 链表

链表是有序的列表，它在内存中的存储形式是以节点的形式存储！

每个节点包括data域和指针域，data域存储数据，指针域存放指向，因此链表不一定是占用连续的内存空间！

链表分为带头结点和没有头结点的链表，根据实际需求确定

链表的初始化：设置一个头结点，头结点固定不变作为该链表的指向。

单向链表：链表只有一个方向，只能单向读取！

实例：D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\LinkedList

双向链表：链表中的每一个节点都有两个指针，指向自己的上一个和下一个节点。头结点上一个节点为null。

实例：D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\DoubleLinkedListDemo

单循环链表：单向链表构成环

实例：

D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\LinkedList\SingleCircleListDemo

约瑟夫问题：一堆人围成一圈，从k开始数数，数到m的人出列，出列后剩下的人重复该步骤直到所有人都出列。问出列的序列。解决方案：单循环链表！

约瑟夫问题思路：遍历循环！

### 栈Stack

栈是一个先入后出的有序列表，栈限制了数据的读写只能在栈的一端进行操作，允许读写的一端为栈顶，不允许的一端为栈底。根据该定义可知，最先入栈的元素在栈底。

栈的应用：计算机计算数学表达式的底层原理！

计算器思路：

创建两个栈，数值栈，符号栈。

对于一个数学表达式，扫描该表达式，如果发现是数字（可能是多位数）就入数值栈

如果发现是操作符，分以下两种情况

1. 如果符号栈是空栈，直接入栈
2. 符号栈中有操作符，将栈中操作符与当前操作符比较优先级，如果当前操作符优先级小于或者等于栈中的操作符，就需要从数栈中出栈两个数，符号栈出一个符号进行运算，得出的结果入数值栈，当前操作符继续和符号栈数据比较优先级直到符号栈为空或者比栈顶优先级高！
3. 表达式扫描完毕，就顺序从两个栈中弹出相应的符号和数组进行运算，得出的结果入数值栈
4. 最后数值栈只有一个数字，那就是结果。

如果数值栈中弹出的两个数据第一个a，第二个是b,符号栈弹出\*,则计算结果是b\*a,底层数据在前，上层数据在后。而符号栈由底向上操作符优先级逐渐增大。不满足的则将当前栈顶的符号出栈运算。

左右括号问题，如果是左括号，入栈，遇到右括号入栈，把左括号之前的数据和操作符按上述规律出栈运算，得出的结果入数值栈。左括号和右括号之间没有任何操作符，左括号和右括号全部出栈。

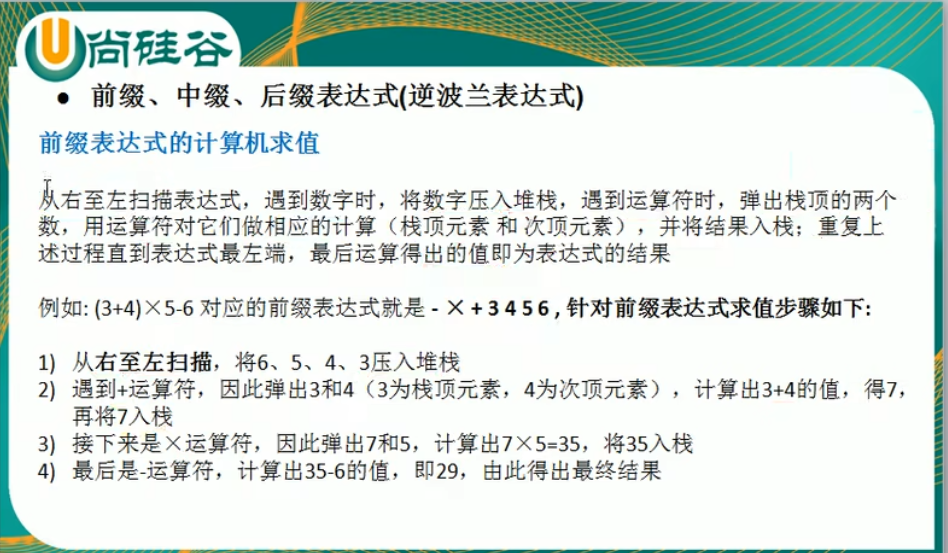
简易计算器：不带括号版本，只能支持1-9以内数值（扫描数有问题）

D:\idea-workspace\JavaSE\src\com\study\sdm\datastru\stack\ArrayStackDemo.java

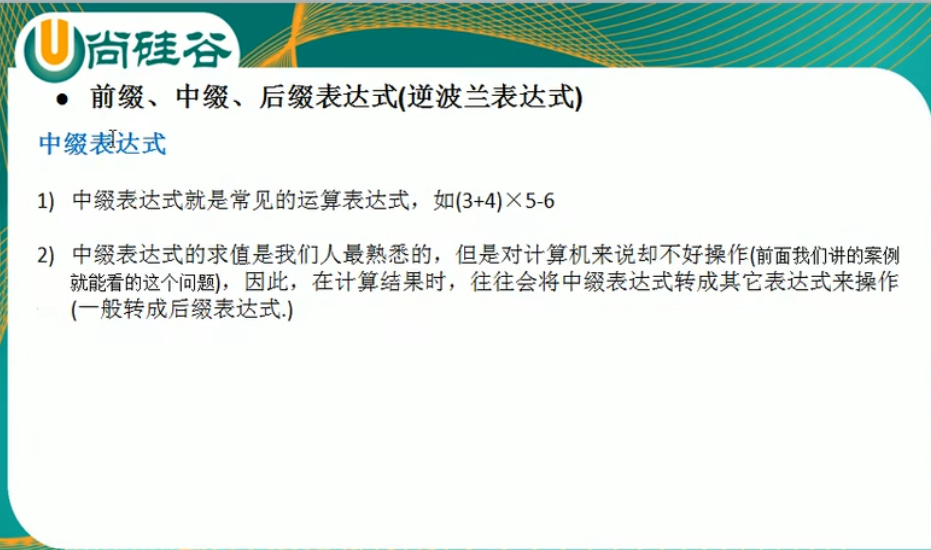
改进思路：解析表达式，得到全部完整的数值和运算符，二一入栈！

### 前缀中缀后缀表达式

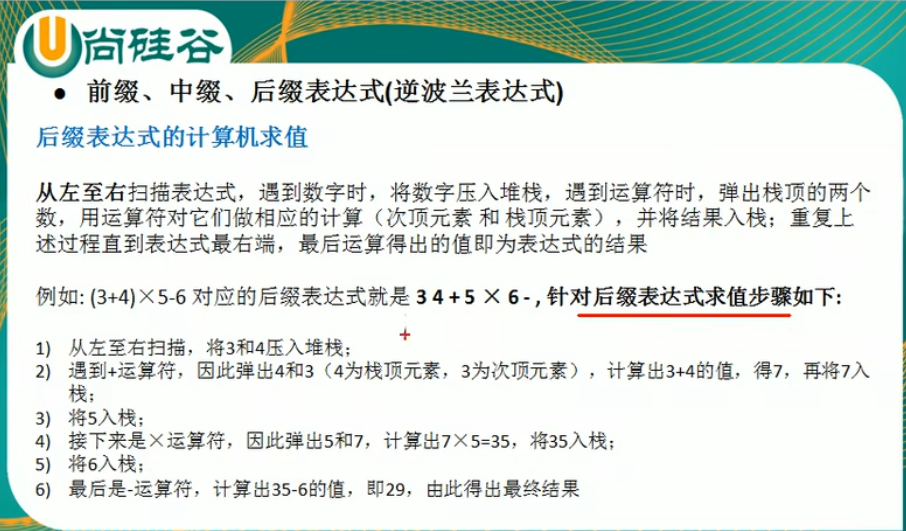
前缀表达式



中缀表达式



后缀表达式



如上，通过后缀表达式（逆波兰表达式）可以大大简化程序运算流程！

逆波兰表达式描述：将操作符放在两个运算对象后面

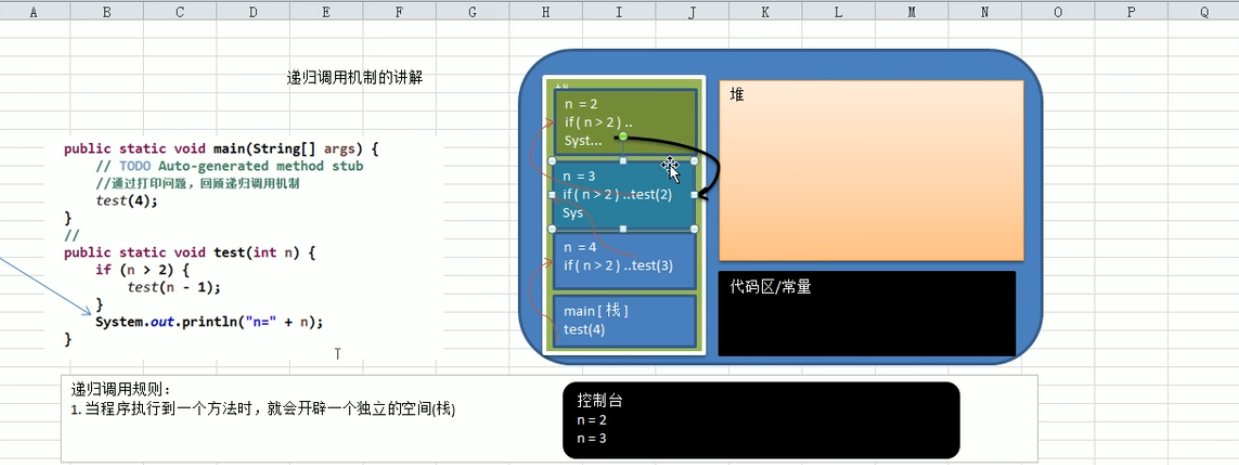
（3+4）\*5-6 对应的逆波兰表达式为34+5\*6-，解析：左右括号直接忽略

3和4两个操作符，后面跟一个操作符，计算后又是一个操作数，入栈，5入栈，这就是两个操作数，后面跟一个乘号，计算结果入栈，跟一个6，又是两个操作数，跟一个减法！

### 递归思想

递归思想：把一个问题分解成规模更小，具有和原问题相同解法的问题。

如图，一个简单的递归调用，test(4)调用test(3)一直到test(2)出if语句，test2执行完毕后再执行栈下方的语句。



递归的应用：

* 用于解决各种数学问题，如八皇后问题，汉诺塔，阶乘问题，迷宫问题，球和篮子问题
* 各类算法也会应用到递归，如快速排序，归并排序，二分查找，分治算法等
* 栈的递归调用

递归的使用规则：

* 执行一个方法时，就创建一个新的受保护的独立空间
* 方法的局部变量是独立的，不会相互影响
* 递归必须向着退出条件去递归，否则就是死归（StackOverflow异常，栈溢出）
* 当一个方法执行完毕或者遇到return返回，遵守谁调用，就将结果返回给谁。当方法执行完毕或者返回时，该方法也就执行完毕。