## 数据结构

数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。

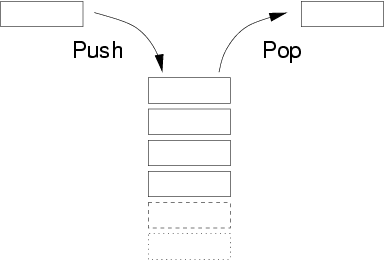
### 线性表

线性linear，指量与量之间按比例、成直线的关系，在空间和时间上代表规则和光滑的运动，，在数学上可以理解为一阶导数为常数的函数。

非线性non-linear则指不按比例、不成直线的关系，代表不规则的运动和突变，一阶导数不为常数。

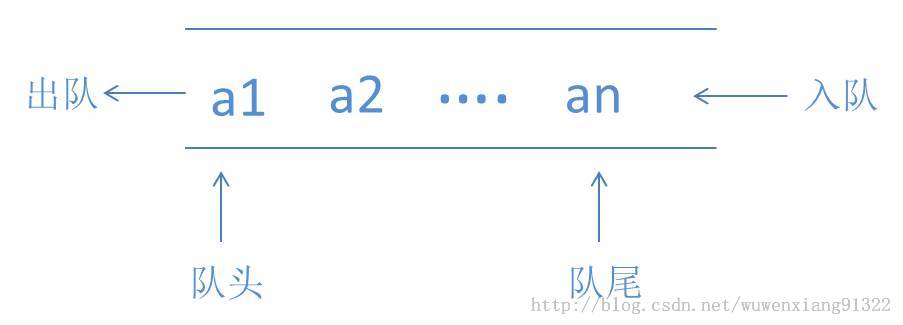
#### 栈(Stack)

栈是一种特殊的线性表，其只能在一个表的一个固定端进行数据节点的插入和删除操作。栈按照后进先出的原则来存储数据，也就是说，先插入的数据将被压入栈底，最后插入的数据在栈顶，读出数据的时候，从栈顶开始逐个读出。栈在汇编语言中，经常用于重要数据的现场保护。栈中没有数据时，称为空栈。



#### 队列(Queue)

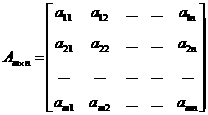
队列和栈类似，也是一种特殊的线性表。和栈不同的是，队列只允许在表的一端进行插入操作，而在另一端进行删除操作，先进先出。一般来说，进行插入操作的一端称为队尾，进行删除操作的一端称为队头。队列中没有元素时，称为空队列。



### 数组(Array)

数组时一种聚合数据类型，是将具有相同类型的若干变量有序地组织在一起的集合。数组是最基本的数据结构，在各种语言中都有对应。按照数据元素的类型，数组可以分为整型数组、字符型数组、浮点型数组、对象数组等。

数组是定长线性表在维数上的扩张，即线性表中的元素又是一个线性表，n维数组是一种“同构”的数据结构，其中每个数据元素类型相同，结构一致。



#### 线性表和数组有什么区别？

相同点：从定义上来看，线性表和数组都是数据元素的有序集.

不同点：

1、数组有维度（比如三维数组）的概念而线性表没有.

2、数组和线性表上可进行的操作不一样。一般我们不在数组上进行数据插入和删除的操作，同样，我们也无法直接通过数据序列来访问线性表中的数据单元（比如表中第i个元素）.

3、线性表是一种概念，一种方法。我们最开始定义数组，是比较死的。不能任意改变大小，而线性表则能.

#### 什么是一阶导数 二阶导数?

对原函数求导数，得到计算原函数上每一点的斜率的新函数---导函数，简称一

次导数。一次导数可以用来寻找原函数上的极值点的位置。

对一次导函数求导，得到二次导函数。平时所说的导数其实都是指一次导函数。

二次导函数的意义在于判断原函数上每一点的凹凸性，判断极值的特性，极大

还是极小。

一阶导数的几何意义表示曲线上该点切线的角度，二阶导数表示一阶导数的一阶导数.

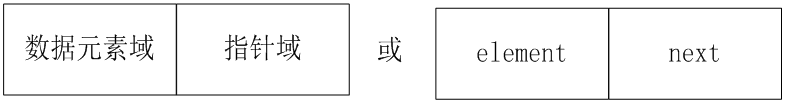
### 链表(Linked List)

链表是一种数据元素按照链式存储结构进行存储的数据结构，在物理存储结构上非连续,逻辑上连续,大小不固定。链表由一系列数据节点构成，每个数据节点包括数据域和引用域两部分。其中，引用域保存了数据结构中下一个元素存放的地址。链表结构中数据元素的逻辑顺序是通过链表中的引用链接次序来实现的。

数组是连续的一片内存，链表是不连续的内存。

链表：单向链表、双向链表

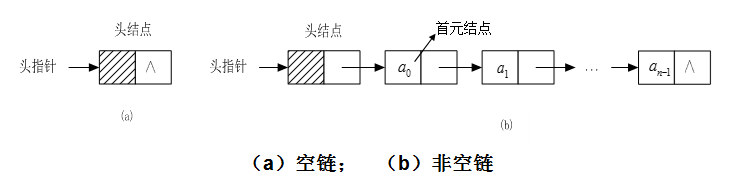
单链表中每个结点的结构：

[](http://images0.cnblogs.com/blog/641601/201508/262042454228505.png)

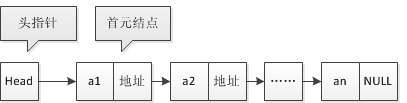
单链表有带头结点和不带头结点结构两种。

头指针所指的不存放数据元素的第一个结点称作**头结点**（头结点指向首元结点）。头结点的数据域一般不放数据（当然有些情况下也可存放链表的长度、用做监视哨等）

存放第一个数据元素的结点称作**第一个数据元素结点**，或称**首元结点**。

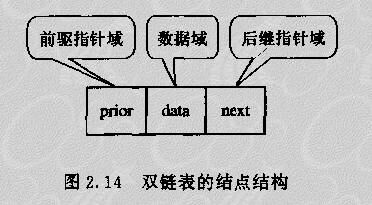


不带头结点的单链表如下：

[](http://images0.cnblogs.com/blog/641601/201508/262042480313734.jpg)

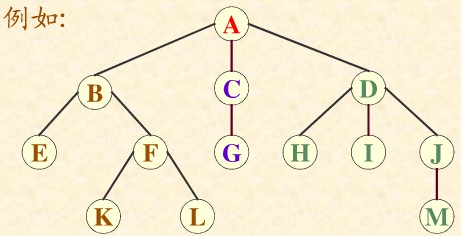
**双向链表**也叫**双链表**，是链表的一种，它的每个数据结点中都有两个指针，分别指向直接后继next和直接前驱prev。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点。为了标识链表的头和尾，将第一个元素的prev指针和最后一个元素的next指针设置为NULL

要反向遍历整个双向链表，使用prev指针从尾到头的顺序访问各个元素，因此为每个元素增加了一个指针的代价，换来的是双向链表更加灵活的访问。

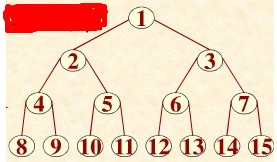


### 树(Tree)

树是典型的非线性结构，其是包括n个节点的有穷结合K。在树结构中，有且仅有一个根节点，该节点没有前驱节点。在树结构中的其他节点有且仅有一个前驱节点，而且可以有m个后继节点，m≧0。

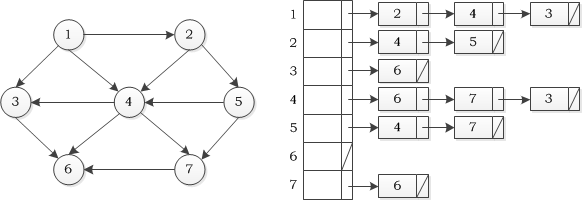


二叉树又叫二分树，它的特点是每个结点最多只有二棵子树，也就是二叉树中没有度大于2的结点。二叉树的子树有左右之分，严格区分左孩子、右孩子，其次序不能颠倒。



### 图(Graph)

图是另外一种非线性数据结构。在图结构中，数据节点一般称为顶点，而边是顶点的有序偶对。如果两个顶点之间存在一条边，那么就表示这两个顶点具有相邻关系。



**图的邻接表表示**

### 堆(Heap)

堆（也叫优先队列），是一棵完全二叉树，它的特点是父节点的值大于（小于）两个子节点的值（分别称为大顶堆和小顶堆）。它常用于管理算法执行过程中的信息，应用场景包括堆排序，优先队列等。

### 散列表

数组的特点是：寻址容易，插入和删除困难；而链表的特点是：寻址困难，插入和删除容易。那么我们能不能综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构？答案是肯定的，这就是我们要提起的哈希表.也被称作散列表。

散列表(hash table)是实现字典操作的一种有效的数据结构,源自于散列函数(Hash function)，其思想是如果在结构中存在关键字和T相等的记录，那么必定在F(T)的存储位置可以找到该记录，这样就可以不用比较而直接取得所查记录。

参考：<http://www.cppblog.com/guogangj/archive/2009/10/15/98699.html>

## 顺序存储、链接存储

**顺序存储**表示是将数据元素存放于一个连续的存储空间中，实现顺序存取或(按下标)直接存取。它的存储效率高，存取速度快。但它的空间大小一经定义，在程序整个运行期间不会发生改变，因此，不易扩充。同时，由于**在插入或删除时，为保持原有次序(**没有规定元素进栈顺序**)，平均需要移动一半(或近一半)元素**，修改效率不高。

**链接存储**表示的存储空间一般在程序的运行过程中动态分配和释放，且只要存储器中还有空间，就不会产生存储溢出的问题。同时在插入和删除时不需要保持数据元素原来的**物理顺序**，只需要保持原来的**逻辑顺序**，因此不必移动数据，只需修改它们的链接指针，修改效率较高。但存取表中的数据元素时，只能循链顺序访问，因此存取效率不高。

### 顺序表和链表的时间性能比较

所谓时间性能是指实现基于这种存储结构的基本运算（即算法）的时间复杂度。

作为一般规律，若线性表需频繁查找却很少进行插入和删除操作，或其操作和“数据元素在线性表中的位置”密切相关时，宜采用顺序表作为存储结构；若线性表需频繁进行插入和删除操作时，则宜采用链表做存储结构。

### 顺序表和链表的空间性能比较

所谓空间性能是指这种存储结构所占用的存储空间的大小。

首先定义结点的 **存储密度**。

顺序表中每个元素的存储密度为 1 ，没有浪费空间；而链表的每个结点除了存放数据元素，还要附加一个指示元素之间逻辑关系的指针，如果数据域占据的空间较小，则链表的结构性开销就占去了整个存储空间的大部分，因而从结点的存储密度上讲，顺序表的存储空间利用率较高。

由于顺序表需要预分配一定长度的存储空间，如果事先不能明确知道线性表的大致长度，则有可能对存储空间预分配得过大，致使在程序执行过程中很大一部分的存储空间得不到充分利用，而造成浪费；若估计得过小，又将造成频繁地进行存储空间的再分配。而链表的显著优点之一就是其存储分配的灵活性，不需要为链表预分配空间，只要有可用的内存空间分配，链表中的元素个数就没有限制。

作为一般规律，当线性表中元素个数变化较大或者未知时，最好使用链表实现；而如果用户事先知道线性表的大致长度，使用顺序表的空间效率会更高。

**总之，**线性表的顺序实现和链表实现各有其优缺点，不能笼统地说哪种实现更好，只能根据实际问题的具体需要，并对各方面的优缺点加以综合平衡，才能最终选定比较适宜的实现方法。

## java体现方式

### ArrayList

ArrayList其实就是一组长度可变的数组，当实例化了一个ArrayList，该数据也被实例化了，当向集合中添加对象时，数组的大小也随着改变，这样它所带来的有优点是快速的随机访问，即使访问每个元素所带来的性能问题也是很小的，但缺点就是想其中添加或删除对象速度慢，当你创建的数组是不确定其容量，所以当我们改变这个数组时就必须在内存中做很多的处理，如你想要数组中任意两个元素中间添加对象，那么在内存中数组要移动所有后面的对象。

### LinkedList

LinkedList是通过节点的连接实现双向链表的数据结构，向linkedList中插入或删除元素的速度是特别快，而随机访问的速度相对较慢，这个是由于链表本身的性质造成的，在链表中，每个节点都包含了前一个节点的引用，后一个节点的引用和节点存储值，当一个新节点插入式，只需要修改其中相关的前后关系节点引用即可，删除节点也是一样。操作对象只需要改变节点的链接，新节点可以存放在内存的任何位置，但也就是因为如此LinkedList虽然存在get()方法，但是这个方法通过遍历节点来定位所以速度很慢。

LinkedList实现了List接口，允许null元素，LinkedList还单独具

addFrist(),addLast(),getFrist(),getLast(),removeFirst(),removeLast()方法，这些方法操作在LinkedList的首部或尾部，这些方法使得LinkedList可以作为堆栈（stack），队列(queue)，和双向队列(deque)来使用。

注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：

List list = Collections.synchronizedList(new LinkedList(...));

### Stack

Stack继承自Vector，实现一个后进先出的堆栈。Stack提供5个额外的方法使得Vector得以被当作堆栈使用。基本的push和pop方法，还有peek方法得到栈顶的元素，empty方法测试堆栈是否为空，search方法检测一个元素在堆栈中的位置。Stack刚创建后是空栈。

Hashtable通过initial capacity和load factor两个参数调整性能。通常缺省的load factor 0.75较好地实现了时间和空间的均衡。增大load factor可以节省空间但相应的查找时间将增大，这会影响像get和put这样的操作 。

总结：

如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。

如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，其效率较高，如果多个线程可能同时操作一个类，应该使用同步的类。

要特别注意对哈希表的操作，作为key的对象要正确复写equals和hashCode方法。

尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。这就是针对抽象编程。