数据结构大体成上可以分成两种:

1. 线性结构

2. 非线性结构(树,图)

线性结构分两种

连续存储 (数组)

也就是指每1个节点在物理内存上是相连的

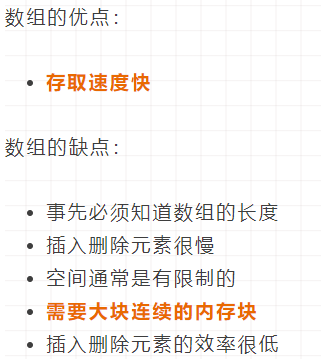
离散存储(链表)

节点在物理内存上并不一定相连, 而是利用指针来关联

数组和链表是线性存储结构的基础，栈和队列是线性存储结构的应用

**数组**

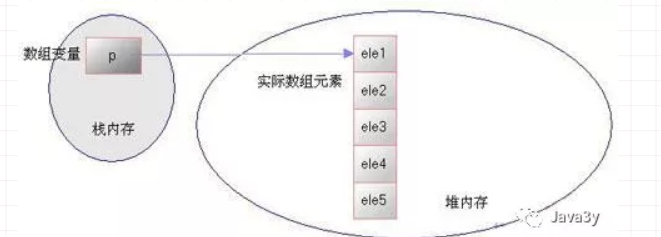
**数组是一种连续存储线性结构，元素类型相同，大小相等**

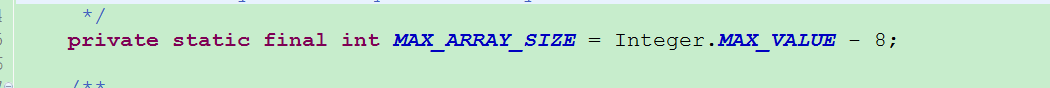


数据类型[] 数组名称 = new 数据类型[长度];//长度要写，不然提示错误

ClassArrayDemo[] t = new ClassArrayDemo[3];

t[0] = new ClassArrayDemo(18, "zs");





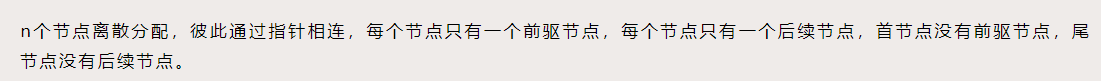
数组的对象头里有一个\_length字段，获取数组长度，只需要去读\_length字段就可以了

所以ArrayList中定义的最大长度为Integer最大值减8，这个8用来数组\_length字段

**链表**

**链表是离散存储线性结构**

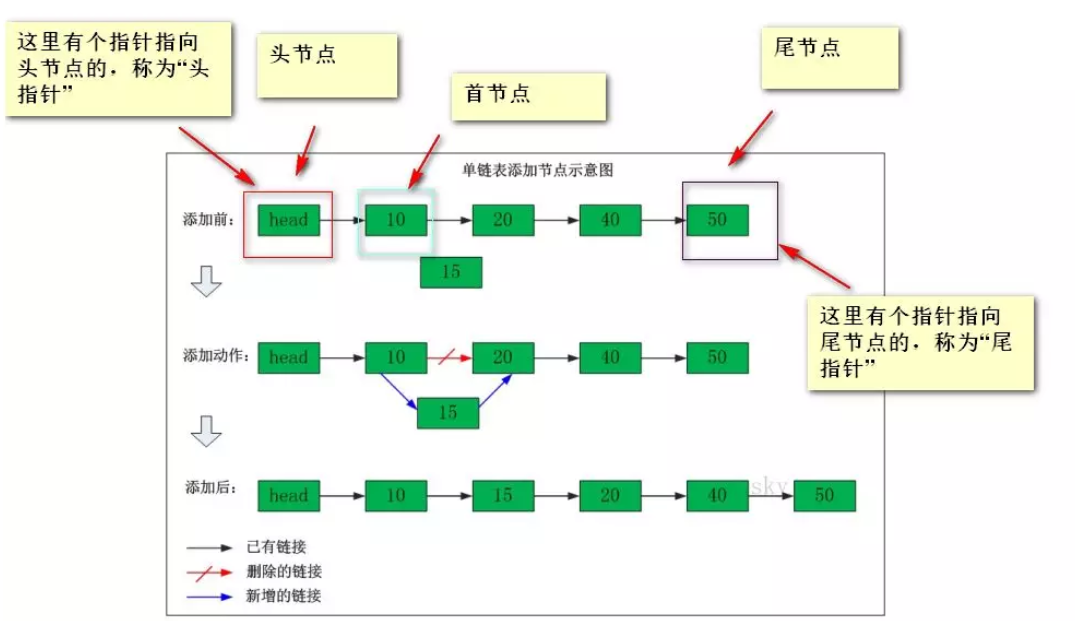
（将内存中零散的内存单元通过引用关联起来，形成一个可以按序号索引的线性结构，与数组的连续存储方式相比，内存的利用率更高）



***离散分配：***

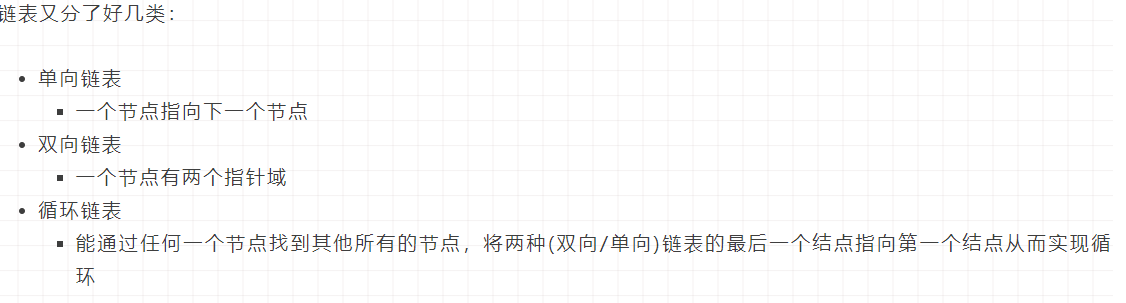
***作业按规定大小分成小份；内存也按同样大小划分成小份***

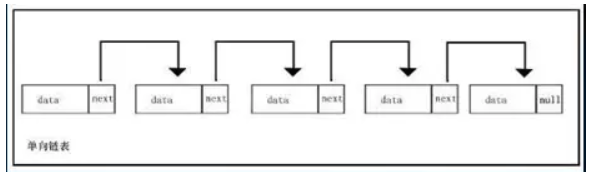
***作业任意一小份可分散放入内存任意未使用的小份***

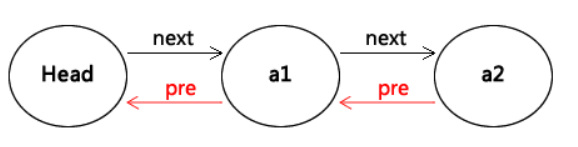


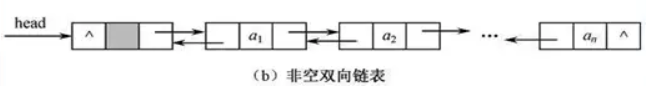
//读写

**确定一个链表我们只需要头指针**









**双链优点：**

* 单链增删节点简单，双链复杂; 需要多分配一个指针存储空间
* 只能从头到尾遍历; 可进可退

**Hash**

散列定义：把任意长度的输入通过散列算法，变换成固定长度的输出，该输出就是散列值（哈希值）

根据散列函数和解决冲突的方法将一组关键字映射到一个有限的连续的地址集（区间）上，把散列值作为记录/元素在表中的存储位置， 这一映象过程称为散列造表或散列

*Hash的底层是散列表，而在Java中散列表的实现是通过数组+链表*

键key用来标识一个对象

散列函数(有多种)：把key转化为数组下标的映射方法

散列值（数组下标）：通过散列函数计算得到的值，也是key对应的value在散列表存储的位置

散列表通过散列函数把元素的键值映射为数组下标，然后将数据存储在数组对应下标的位置

(数组的长度即哈希表的长度)

Java的 hashCode（）方法：根据一定的规则将与对象相关的信息（比如对象的存储地址，对象的字段等）映射成一个int数值

比较

非哈希表的特点：关键字在表中的位置和它之间不存在一个确定的关系，查找的过程为给定值和各个关键字进行比较，查找的效率取决于和给定值进行比较的次数。

哈希表的特点：关键字在表中位置和它之间存在一种确定的关系。

hash冲突：不同的key通过hash函数得到相同的值

根据解决冲突的方式不同，桶可以有不同的数据结构(单链表/数组元素)

hash表采用开放定址法

1. 遇到冲突的时候就找hash表剩下空余的空间然后插入
2. (底层仍是一个数组，元素存放键对应的值)

H i ( key ) = ( H ( key )+ d i ) mod m ( i = 1,2,…… ， k ( k ≤ m – 1))

H ( key ) 为关键字 key 的直接哈希地址，

m 为哈希表的长度，

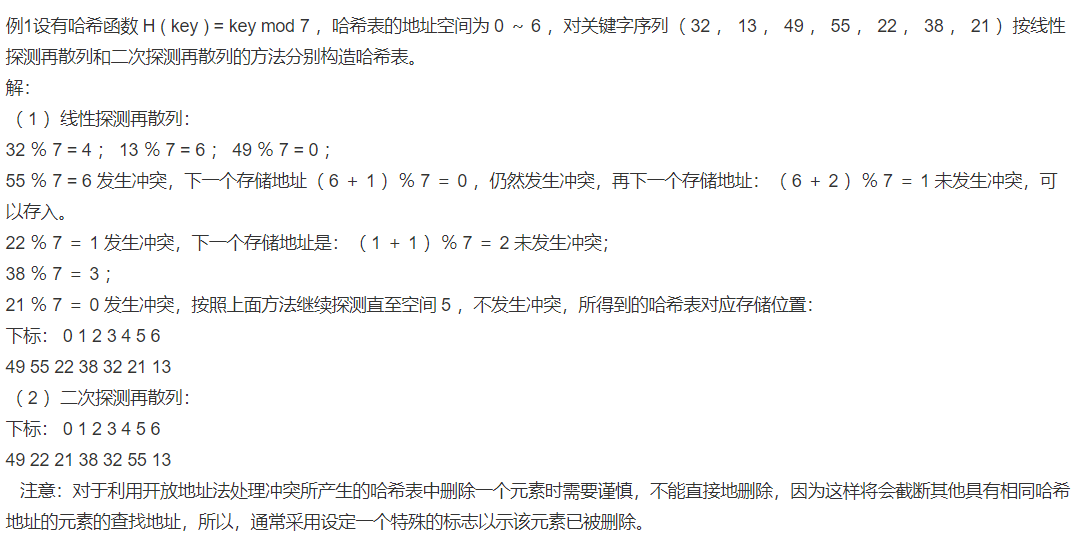
di 为每次再探测时的地址增量

增量 d 可以有不同的取法，并根据其取法有不同的称呼：

（ 1 ） d i ＝ 1 ， 2 ， 3 ， …… 线性探测再散列；

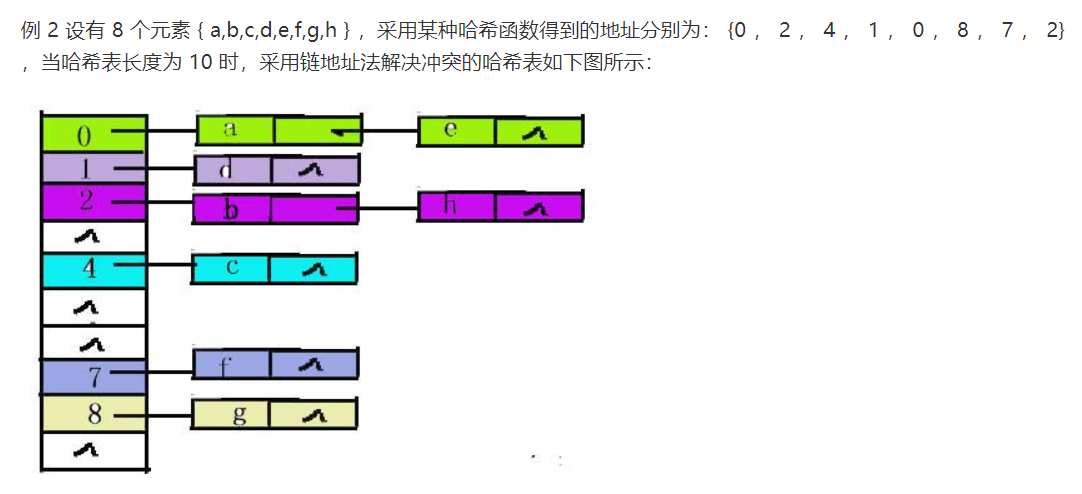
（ 2 ） d i ＝ 1^2 ，－ 1^2 ， 2^2 ，－ 2^2 ， k^2， -k^2…… 二次探测再散列；

（ 3 ） d i ＝ 伪随机序列 伪随机再散列



Hash表采用链地址法

1. 将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个元素中，查找、插入和删除主要在同义词链中进行
2. 链地址法适用于经常进行插入和删除的情况



如果散列表太满，则需要对散列表再散列，创建一个桶数更多的散列表，并将原有的元素插入到新表中，丢弃原来的表

装填因子(load factor) a=n/m 其中n 为关键字个数，m为表长决定了何时对散列表再散列

装填因子默认为0.75，如果表中超过了75%的位置已经填入了元素，那么这个表就会用双倍的桶数自动进行再散列

**树**

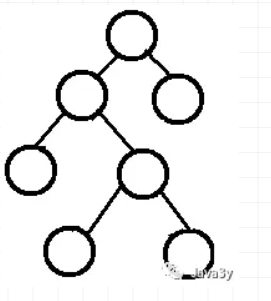
* 树是由结点或顶点和边组成的，没有结点的树称为空(null或empty)树
* 一个结点和另一个结点之间的连接被称之为边
* 结点所拥有的子树个数称为结点的度
* 结点的层次(Level)从根(Root)开始定义起，根为第0层，根的孩子为第1层

二叉树：

每个结点至多拥有两棵子树，并且，二叉树的子树有左右之分，其次序不能任意颠倒

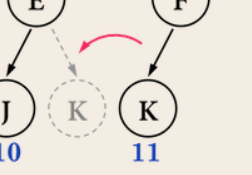
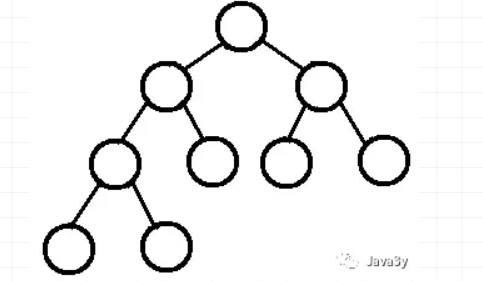
完满二叉树：

除了叶子结点之外的每一个结点都有两个孩子结点



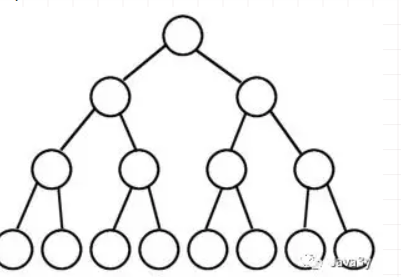
完全二叉树：

除了最后一层之外的其他每一层都被完全填充，并且所有结点都保持向左对齐



满二叉树：

除了叶子结点之外的每一个结点都有两个孩子，每一层(当然包含最后一层)都被完全填充



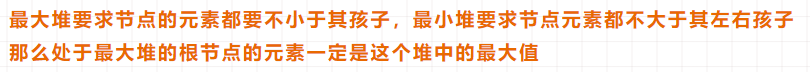
堆排序是将数据看成是完全二叉树、根据完全二叉树的特性来进行排序的一种算法

完全二叉树特性：

左边子节点位置 = 当前父节点的两倍 + 1

右边子节点位置 = 当前父节点的两倍 + 2





最大堆：节点元素!<其子，根节点即最大值

最小堆：节点元素!>其子

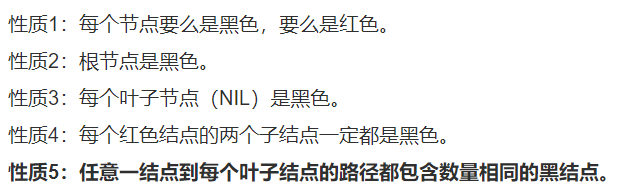
平衡二叉树特点：

* 非叶子节点最多拥有两个子节点；
* 非叶子节值大于左边子节点、小于右边子节点；
* 树的左右两边的层级数相差不会大于1;
* 没有值相等重复的节点;

B/B+树: 用在磁盘文件组织 数据索引和数据库索引

红黑树是一种含有红黑结点并能自平衡的二叉查找树

可以保证二叉树基本符合均衡的结构(不会瘸腿)



红黑树能自平衡，它靠的是什么？三种操作：左旋、右旋和变色