- 哈希表
  - HashMap（jdk）
    - 基本概念
      - HashMap 是一个散列桶（数组和链表），它存储的内容是键值对 key-value 映射
      - HashMap 采用了数组和链表的数据结构，能在查询和修改方便继承了数组的线性查找和链表的寻址修改
      - HashMap 是非 synchronized，所以 HashMap 很快
      - HashMap 可以接受 null 键和值，而 Hashtable 则不能（原因就是 equlas() 方法需要对象，因为 HashMap 是后出的 API 经过处理才可以）
      - 查找效率主要取决于散列函数、处理冲突的方法和装载因子
    - 具体的 put 过程（JDK1.8）
      - 对 Key 求 Hash 值，然后再计算下标
      - 如果没有碰撞，直接放入桶中（碰撞的意思是计算得到的 Hash 值相同，需要放到同一个 bucket 中）
      - 如果碰撞了，以链表的方式链接到后面
      - 如果链表长度超过阀值（TREEIFY THRESHOLD==8），就把链表转成红黑树，链表长度低于 6，就把红黑树转回链表
      - 如果节点已经存在就替换旧值
      - 如果桶满了（容量16*加载因子0.75），就需要 resize（扩容2倍后重排）
    - HashMap 与 HashTable 区别
      - 默认容量不同，扩容不同
      - 线程安全性：HashTable 安全
      - 效率不同：HashTable 要慢，因为加锁
  - 常用的散列函数
    - 直接寻址法：取关键字或关键字的某个线性函数值为散列地址。即H(key)=key或H(key) = a · key + b，其中a和b为常数（这种散列函数叫做自身函数）。若其中H(key）中已经有值了，就往下一个找，直到H(key）中没有值了，就放进去
    - 数字分析法：分析一组数据，比如一组员工的出生年月日，这时我们发现出生年月日的前几位数字大体相同，这样的话，出现冲突的几率就会很大，但是我们发现年月日的后几位表示月份和具体日期的数字差别很大，如果用后面的数字来构成散列地址，则冲突的几率会明显降低。因此数字分析法就是找出数字的规律，尽可能利用这些数据来构造冲突几率较低的散列地址
    - 平方取中法：当无法确定关键字中哪几位分布较均匀时，可以先求出关键字的平方值，然后按需要取平方值的中间几位作为哈希地址。这是因为：平方后中间几位和关键字中每一位都相关，故不同关键字会以较高的概率产生不同的哈希地址
    - 折叠法：将关键字分割成位数相同的几部分，最后一部分位数可以不同，然后取这几部分的叠加和（去除进位）作为散列地址。数位叠加可以有移位叠加和间界叠加两种方法。移位叠加是将分割后的每一部分的最低位对齐，然后相加；间界叠加是从一端向另一端沿分割界来回折叠，然后对齐相加
    - 随机数法：选择一随机函数，取关键字的随机值作为散列地址，通常用于关键字长度不同的场合
    - 除留余数法：取关键字被某个不大于散列表表长m的数p除后所得的余数为散列地址。即 H(key) = key MOD p,p<=m。不仅可以对关键字直接取模，也可在折叠、平方取中等运算之后取模。对p的选择很重要，一般取素数或m，若p选的不好，容易产生同义词
    - 乘法取整法：f(x):=trunc((x/maxX)*maxlongit) mod maxM，主要用于实数
  - 解决 hash 碰撞
    - 开放定址法：Hi=(H(key) + di) MOD m,i=1,2，…，k(k<=m-1），其中H(key）为散列函数，m为散列表长，di为增量序列，可有下列三种取法
      - 线性探查法
        - 探查过程终止于三种情况
          - 若当前探查的单元为空，则表示查找失败（若是插入则将key写入其中）
          - 若当前探查的单元中含有key，则查找成功，但对于插入意味着失败
          - 若探查到T[d-1]时仍未发现空单元也未找到key，则无论是查找还是插入均意味着失败(此时表满)
        - 缺点
          - 处理溢出需另编程序。一般可另外设立一个溢出表，专门用来存放上述哈希表中放不下的记录。此溢出表最简单的结构是顺序表，查找方法可用顺序查找
          - 按上述算法建立起来的哈希表，删除工作非常困难。假如要从哈希表 HT 中删除一个记录，按理应将这个记录所在位置置为空，但我们不能这样做，而只能标上已被删除的标记，否则，将会影响以后的查找
          - 线性探测法很容易产生堆聚现象。所谓堆聚现象，就是存入哈希表的记录在表中连成一片。按照线性探测法处理冲突，如果生成哈希地址的连续序列愈长（即不同关键字值的哈希地址相邻在一起愈长），则当新的记录加入该表时，与这个序列发生冲突的可能性愈大。因此，哈希地址的较长连续序列比较短连续序列生长得快，这就意味着，一旦出现堆聚（伴随着冲突），就将引起进一步的堆聚
      - 二次探查法：di=1^2,-1^2,2^2,-2^2，(3)^2，…，±（k)^2,(k<=m/2)
      - 伪随机探测再散列：di=伪随机数序列
    - 再散列法：Hi=RHi(key),i=1,2，…，k RHi均是不同的散列函数，即在同义词产生地址冲突时计算另一个散列函数地址，直到冲突不再发生，这种方法不易产生“聚集”，但增加了计算时间
    - 链地址法（拉链法）
    - 建立一个公共溢出区