HashMap

loadFactor负载因子默认值为什么是0.75？

0.75是对空间和时间效率的一个平衡。

如果因子很高，数组利用率会很高，但发生碰撞几率也高。查询效率会降低

如果因子很低，数组利用率会很低，发生碰撞几率也低。比较浪费空间

数组长度取值2次幂的原因

* 计算元素存储位置时hash & (length-1)，可以用&代替%操作，提升性能
* 数组扩容时，仅仅关注 “特殊位” 就可以重新定位元素

为什么重新计算hash

目标：针对小数组的优化，减少碰撞（hash值是int类型，取值范围几十亿，小数组定位时只用到了hash的低位）

计算过程：将原hash值中高16位与低16位异或运算，使hash值低位中同时含有高位信息

为什么要引入红黑树？

数据量大的情况下，为了提高链表查找性能

线程安全问题

添加数据时，数据可能会丢失

死循环（jdk8已经解决，但是tree可能出现类似问题）

1.7扩容迁移元素时，会将元素顺序改变，导致两个线程中出现元素相互指向

Put()源码解析

1.如果map是空的，初始化map（创建长度默认为16的数组）

2.计算键值key的hash值并确定元素要存放在数组中的位置

3.插入到链表或红黑树中。如果key存在，直接覆盖；如果链表长度大于8并且数组长度大于等于64，转化为红黑树，数组长度小于 64，直接扩容

4.判断元素个数是否超过临界值threshold，如果超过，进行扩容

扩容源码解析

1.新建数组。如果map为空，初始化map；否则容量扩大一倍

2.将元素移动到新数组中

(1)该位置只有一个元素，直接移动

(2)如果是链表，移动到新数组还是链表

(3)如果是树，移动后如果节点个数小于等于6，退化为链表

ArrayList,Vector, LinkedList

ArrayList和Vector都是使用数组方式存储数据，Vector是线程安全的。

LinkedList底层是链表，添加数据较快

ArrayList

不带参数的构造方法生成对象时，生成一个长度为10的Object类型数组。

复制方式来扩充容量，扩充1.5倍。

System.arraycopy为C++编写的底层函数，直接对内存中的数据块进行复制的，是一整块一起复制的而采用下标一个一个地进行赋值时，时间主要浪费在了寻址和赋值上。

数组是Native的线性序列，所以效率高

ArrayList与LinkedList区别

ArrayList查找效率高

LinkedList添加、删除数据较快

HashSet

基于HashMap，元素存储在HashMap里的key位置。通过hashcode和equals方法去重

ArrayList与HashSet区别

ArrayList元素有序，允许元素重复

HashSet 元素无序，元素不重复