哈希表也称为散列表，是根据关键字值（key value）而直接进行访问的数据结构。也就是说，它通过把关键字值映射到一个位置来访问记录，以加快查找的速度。这个映射函数称为哈希函数（也称为散列函数），映射过程称为哈希化，存放记录的数组叫做散列表。

对不同的关键字，可能得到同一个散列地址，即同一个数组下标，这种现象称为冲突

解决冲突的方法：

开放地址法：

通过系统的方法找到数组的另一个空位，把数据填入，而不再用哈希函数得到的数组下标，因为该位置已经有数据了；

-- 线性探测法：线性地查找空白单元，数组下标一直递增，直到找到空白位

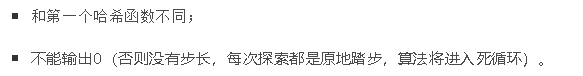
弊端：数据可能会发生聚集。一旦聚集形成，它会变得越来越大，那些哈希化后落在聚集范围内的数据项，都要一步步的移动，并且插在聚集的最后，因此使聚集变得更大。聚集越大，它增长的也越快。这就导致了哈希表的某个部分包含大量的聚集，而另一部分很稀疏。

进行二次探测：二次探测是防止聚集产生的一种方式，思想是探测相隔较远的单元，而不是和原始位置相邻的单元。如果哈希函数计算的原始下标是x, 线性探测就是x+1, x+2, x+3, 以此类推；而在二次探测中，探测的过程是x+1, x+4, x+9, x+16，二次探测虽然消除了原始的聚集问题，但是产生了另一种更细的聚集问题，叫二次聚集：比如讲184，302，420和544依次插入表中，它们的映射都是7，那么302需要以1为步长探测，420需要以4为步长探测， 544需要以9为步长探测。只要有一项其关键字映射到7，就需要更长步长的探测，这个现象叫做二次聚集。

-- 再哈希法（根据不同的key动态获取步长）

再哈希法就是把关键字用不同的哈希函数再做一遍哈希化，用这个结果作为步长，对于指定的关键字，步长在整个探测中是不变的，不同关键字使用不同的步长。

第二个哈希函数必须具备如下特点：





其中 constant 是质数，且小于数组容量。如果数组容量为13, 质数，探测序列最终会访问所有单元。即 0,5,10,2,7,12,4,9,1,6,11,3,一直下去，只要表中有一个空位，就可以探测到它。

链地址法：在哈希表每个单元中设置链表（即链地址法），某个数据项的关键字值还是像通常一样映射到哈希表的单元，而数据项本身插入到这个单元的链表中。其他同样映射到这个位置的数据项只需要加到链表中，不需要在原始的数组中寻找空位。

在没有冲突的情况下，哈希表中执行插入和删除操作可以达到O(1)的时间级，这是相当快的，如果发生冲突了，存取时间就依赖后来的长度，查找或删除时也得挨个判断，但是最差也就O(N)级别。

# HashMap和Hashtable区别？

不同点

继承的类不一样：HashMap继承的AbstractMap抽象类，Hashtable继承的Dictionay抽象类

应对多线程处理方式不一样：HashMap是非线程安全的，Hashtable是线程安全的，所以Hashtable效率比较低

定位算法不一样：HashMap通过key的hashCode()进行hash()得到哈希地址，数组下标=哈希地址 & (容量 - 1)，采用的是与运算，所以容量需要是2的幂次方结果才和取模运算结果一样。而Hashtable则是：数组下标=(key的hashCode() & 0x7FFFFFFF ) % 容量，采用的取模运算，所以容量没要求

键值对规则不一样：HashMap允许键值为null，而Hashtable不允许键值为null

哈希表扩容算法不一样：HashMap的容量扩容按照原来的容量\*2，而Hashtable的容量扩容按照原来的容量\*2+1

容量(capacity)默认值不一样：HashMap的容量默认值为16，而Hashtable的默认值是11

put方法实现不一样：HashMap是将节点插入到链表的尾部，而Hashtable是将节点插入到链表的头部

底层结构不一样：HashMap采用了数组+链表+红黑树，而Hashtable采用数组+链表

为什么HashMap允许null键值呢，而Hashtable不允许null键值呢？这里还得先介绍一下什么是null，我们知道Java语言中有两种类型，一种是基本类型还有一种是引用类型，其实还有一种特殊的类型就是null类型，它不代表一个对象(Object)也不是一个对象(Object)，然后在HashMap和Hashtable对键的操作中使用到了Object类中的equals方法，所以如果在Hashtable中置键值为null的话就可想而知会报错了，但是为什么HashMap可以呢？因为HashMap采用了特殊的方式，将null转为了对象(Object)，具体怎么转的，这里就不深究了。

相同点

实现相同的接口：HashMap和Hashtable均实现了Map接口

负载因子(loadFactor)默认值一样：HashMap和Hashtable的负载因子默认都是0.75

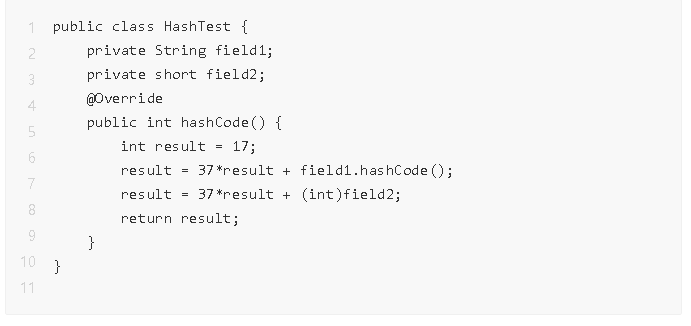
采用相同的方法处理哈希冲突：都是采用链地址法即拉链法处理哈希冲突

相同哈希地址可能分配到不同的链表，同一个链表内节点的哈希地址不一定相同：因为HashMap和Hashtable都会扩容，扩容后容量变化了，相同的哈希地址取到的数组下标也就不一样。

## ****重写 hashCode 方法****

**（1）**

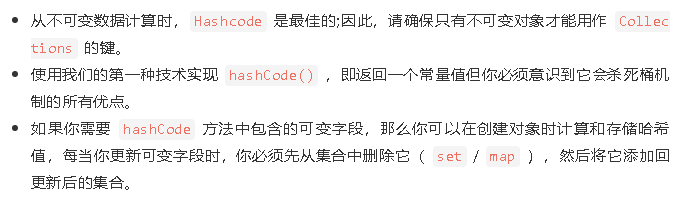




（2） Apache HashCodeBuilder



总结：



内存泄漏

如果未实现 equals() 和 hashcode() ，则 Java 应用程序中可能会发生内存泄漏。HashMap 通过重复添加相同的键而不断增长，最后抛出 OutOfMemoryError 。