详细请看：<https://wenku.baidu.com/view/b5d9a86427d3240c8447efab.html>

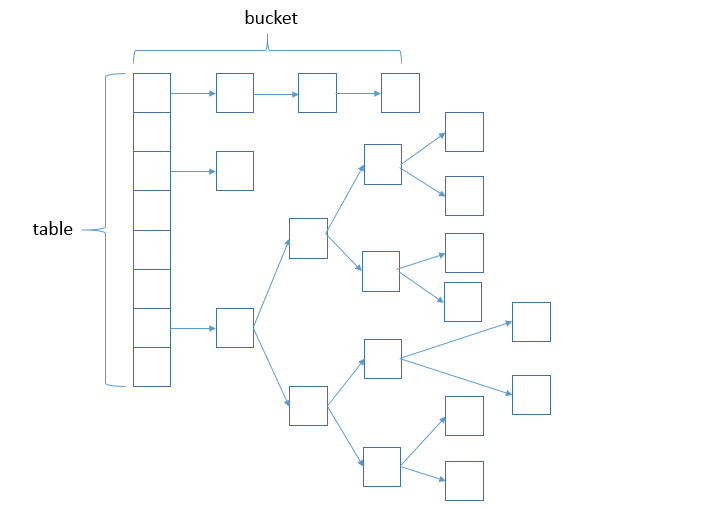
http://www.cnblogs.com/liuyang0/p/6434431.html

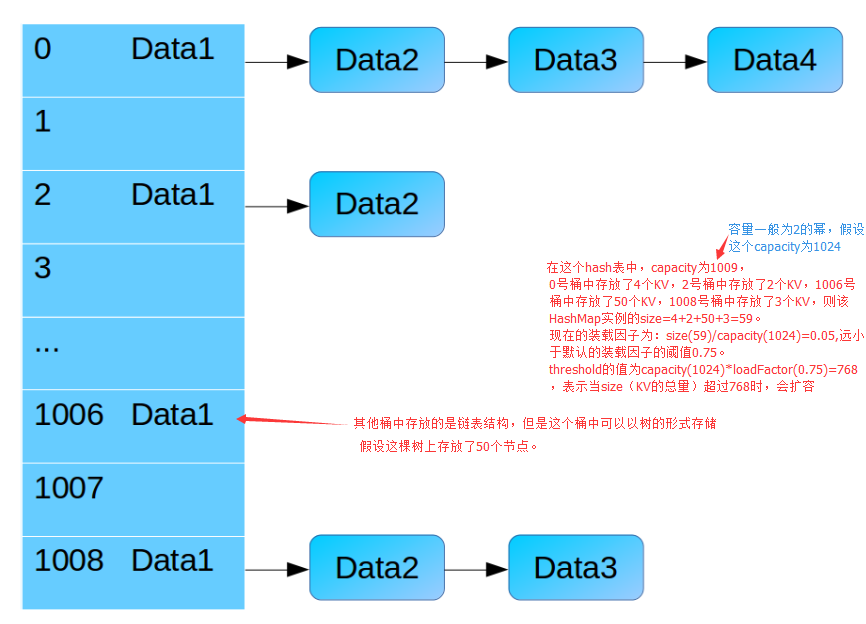
\*\*\*　　JDK1.6中HashMap采用的是位桶+链表的方式，即我们常说的散列链表的方式，而JDK1.8中采用的是位桶+链表/红黑树的方式，也是非线程安全的。当某个位桶的链表的长度达到某个阀值的时候，这个链表就将转换成红黑树。

1. hashmap基于哈希表的 map接口的实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。与hashtable区别是hashmap允许使用null作为key值，且线程不安全，hashtable不允许null作为key值线程安全，可以使用Collections.synchronizedMap使hashmap线程安全,在JDK1.5后建议使用ConcurrentHashMap,ConcurrentHashMap采用锁分段机制，hashtable是一个线程访问时，其他线程无法访问，但ConcurrentHashMap是一个线程访问某个指定数组的某一个下标，该下标所在段不允许其他线程访问，其他线程仍可访问ConcurrentHashMap的其他下标。
2. hashmap底层数据结构是基于数组和链表（红黑树 ）来实现的，它之所以有相当快的查询速度主要是因为它是通过计算散列码来决定存储的位置。通过key值计算hash码值，将其传递进数组，若hash码值冲突则采用链表连接，若该hashmap容量大于64且该链表长度大于8则将该链表转化为红黑树，若链表长度减少到6时候，则将红黑树转化回链表
3. hashmap每次扩容扩大为原来的两倍，但扩容操作需要将数据全部重新计算放入新表，很消耗资源，如果对数据含量有一定预估，建议初始化hashmap时指定容量

1.“你用过HashMap吗？”“什么是HashMap?你为什么用它？”

几乎每个人都会回答是的，然后回答HashMap的一些特性，比如HashMap可以接受null键值和值，而HashTable则不能，HashMap是非synchronized;HashMap很快，以及HashMap存储的是键值对等等。





2.原理

HashMap是基于hashing的原理，我们使用put(key,value)存储对象到HashMap中，使用get(key)从HashMap中获取对象。当我们给put（）方法传递键值和值时，我们先对键调用hashCode（）方法，返回的hashCode用户找到bucket位置来存储Entry对象。

Hash计算下标位置的经典算法 (n-1)&hash n表示hashmap容量

「 (n - 1) & hash] 」定位数组下标位置的经典算法，你可以自己写一个main方法来做几个实验。n=16的话，计算结果肯定在0～15之间。p为计算后取到的一个Node节点 判断是否为null (if else 省略大括号)

**static** **final** **int** hash(Object key) {

**int** h;

**return** (key == **null**) ? 0 : (h = key.hashCode()) ^ (h >>> 16);

}

**size**

size表示HashMap中存放KV的数量（为链表和树中的KV的总和）。

**capacity**

capacity译为容量。capacity就是指HashMap中桶的数量。默认值为16。一般第一次扩容时会扩容到64，之后好像是2倍。总之，**容量都是2的幂**。

/\*\*

\* The default initial capacity - MUST be a power of two.

\*/

static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4; // aka 16

**loadFactor**

loadFactor译为装载因子。装载因子用来衡量HashMap满的程度。loadFactor的默认值为0.75f。计算HashMap的实时装载因子的方法为：size/capacity，而不是占用桶的数量去除以capacity。

/\*\*

\* The load factor used when none specified in constructor.

\*/

static final float DEFAULT\_LOAD\_FACTOR = 0.75f;

**threshold**

threshold表示当HashMap的size大于threshold时会执行resize操作。   
threshold=capacity\*loadFactor

/\*\*

\* The next size value at which to resize (capacity \* load factor).

\*/

int threshold;