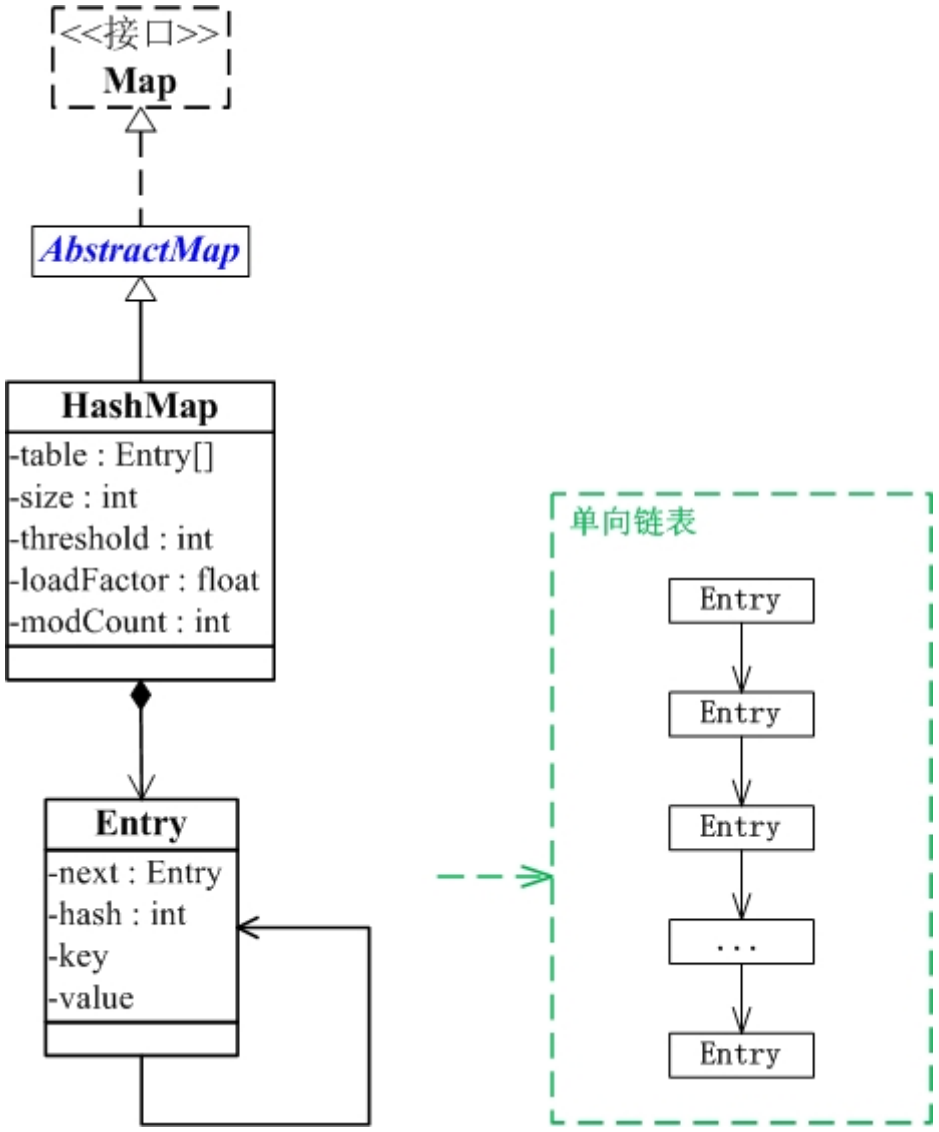
散列表，采用哈希算法基于哈希表实现，所以存储自定义对象作为键时，必须重写hasCode和equals方法。键不能重复，如果发生重复，新的键值对会替换旧的键值对。key、value都可以为null。线程不安全。

JDK1.8 之前 HashMap 由 数组+链表 组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的（“拉链法”解决冲突）.

JDK1.8 以后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）时，将链表转化为红黑树，以减少搜索时间。

继承于AbstractMap，实现了Map、Cloneable、java.io.Serializable接口（串行读取、写入功能）。



table是一个Entry[]数组类型，而Entry实际上就是一个单向链表。哈希表的"key-value键值对"都是存储在Entry数组中的。

size是HashMap保存的键值对的数量。

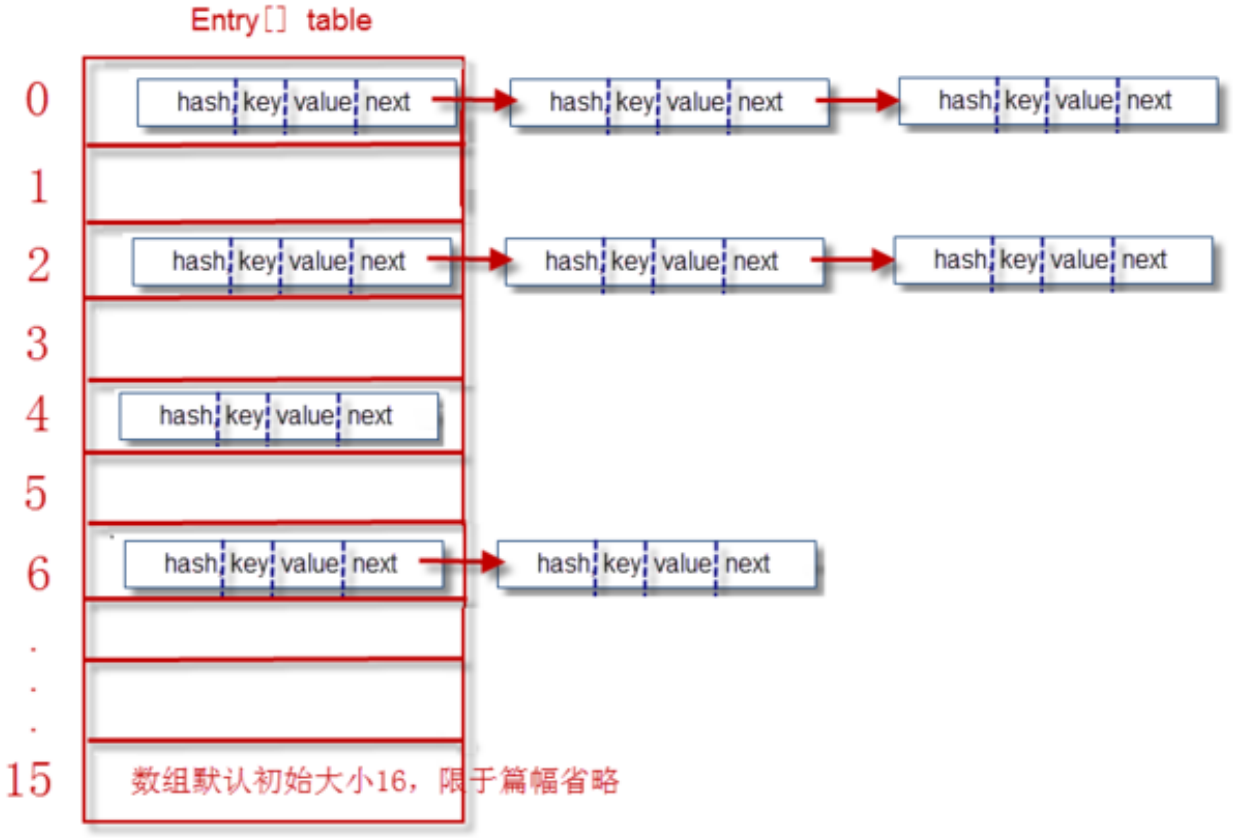
threshold是HashMap的阈值，用于判断是否需要调整HashMap的容量。threshold的值="容量*加载因子"，当HashMap中存储数据的数量达到threshold时，就需要将HashMap的容量加倍。

loadFactor就是加载因子。

modCount是用来实现fail-fast机制的。

存储过程

JDK1.8之前，哈希表的基本结构就是“数组+链表”。**通过拉链法解决哈希冲突。**



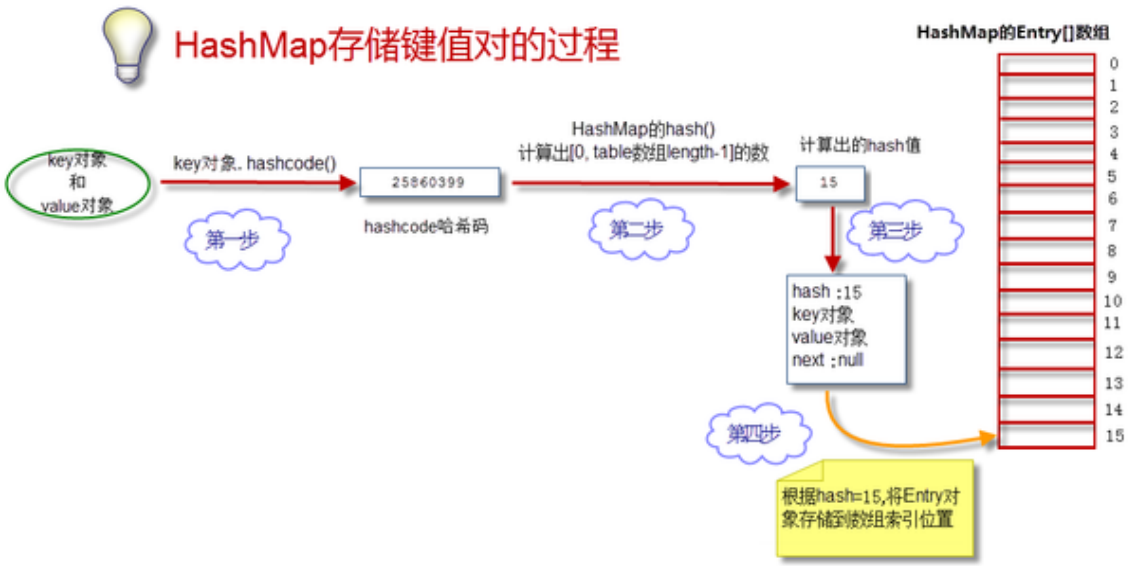


图9-16 HashMap存储数据过程示意图

- **存储数据过程put(key,value)**

(1) 获得key对象的hashcode

首先调用key对象的hashcode()方法，获得hashcode。**注意：**如果是加入HashMap的key是个可变对象，在加入

到集合后又修改key的成员变量的值，可能导致hashCode()值以及equal()的比较结果发生变化，无法访问到该key。一般情况下不要修改。

(2) 根据hashcode计算出hash值(要求在[0, 数组长度-1]区间)

hashcode是一个整数，我们需要将它转化成[0, 数组长度-1]的范围。我们要求转化后的hash值尽量均匀地分布在[0,数组长度-1]这个区间，减少“hash冲突”

```
static int hash(int h) {
    // This function ensures that hashCodes that differ only by
    // constant multiples at each bit position have a bounded
    // number of collisions (approximately 8 at default load factor).
    h ^= (h >>> 20) ^ (h >>> 12);
    return h ^ (h >>> 7) ^ (h >>> 4);
}
static int indexFor(int h, int length) {
    return h & (length-1);
}
```

图9-17 hash算法源码

(3) 生成Entry对象

如上所述，一个Entry对象包含4部分：key对象、value对象、hash值、指向下一个Entry对象的引用。我们现在算出了hash值。下一个Entry对象的引用为null。

(4) 将Entry对象放到table数组中

如果本Entry对象对应的数组索引位置还没有放Entry对象，则直接将Entry对象存储进数组;如果对应索引位置已经有Entry对象，则将已有Entry对象的next指向本Entry对象，形成链表。JDK8中，当链表长度大于8时，链表就转换为红黑树，这样又大大提高了查找的效率。

- 取数据过程get(key)

(1) 获得key的hashcode，通过hash()散列算法得到hash值，进而定位到数组的位置。

(2) 在链表上挨个比较key对象。 调用equals()方法，将key对象和链表上所有节点的key对象进行比较，直到碰到返回true的节点对象为止。

(3) 返回equals()为true的节点对象的value对象。

hashcode()和equals方法的关系：

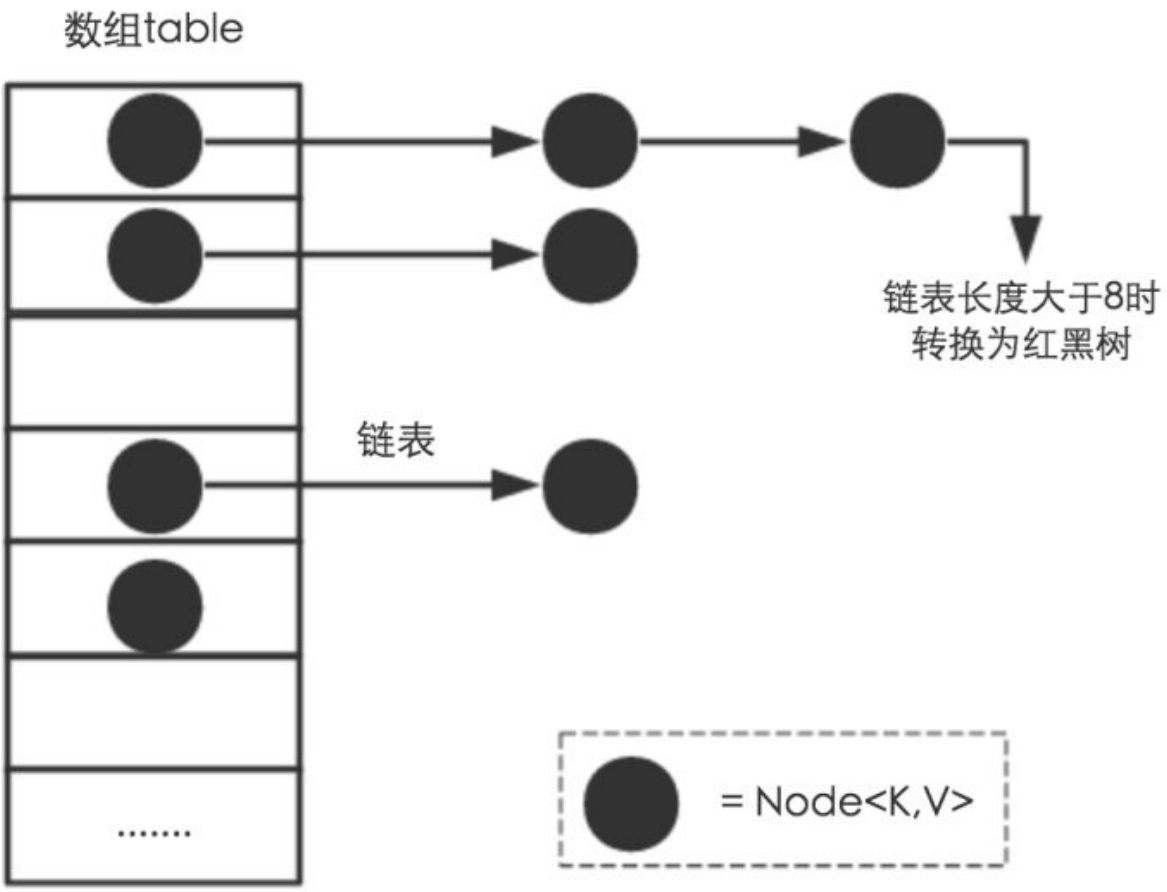
Java中规定，两个内容相同(equals()为true)的对象必须具有相等的hashCode。

- **扩容问题**

HashMap的位桶数组，初始大小为16。如果位桶数组中的元素达到(0.75*数组 length)， 就重新调整数组大小变为原来2倍大小。加载因子和初始容量可设置。

扩容很耗时。扩容的本质是定义新的更大的数组，并将旧数组内容挨个拷贝到新数组中。

JDK1.8之后，解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）时，将链表转化为红黑树，以减少搜索时间。



源码

主要API

clear()

clear() 的作用是**清空HashMap**。它是通过将所有的元素设为null来实现的。

containsKey(key)

containsKey() 的作用是**判断HashMap是否包含key**。

首先*通过getEntry(key)获取key对应的Entry，然后判断该Entry是否为null*。getEntry() 的作用就是**返回“键为key”的键值对**。

注意：**HashMap将“key为null”的元素都放在table的位置0处。**

containsValue(value)

containsValue() 的作用是**判断HashMap是否包含“值为value”的元素。**

分为两步：第一，若“value为null”，则调用containsNullValue()。第二，若“value不为null”，则查找HashMap中是否有值为value的节点。

containsNullValue() 的作用**判断HashMap中是否包含“值为null”的元素。**

entrySet()、values()、keySet()

原理类似，entrySet()的作用是**返回“HashMap中所有Entry的集合”，它是一个集合。**

get(key)

get() 的作用是**获取key对应的value。**

put(K key , V value)

put() 的作用是**对外提供接口，让HashMap对象可以通过put()将“key-value”添加到HashMap中。**key不存在HashMap中时调用

addEntry ( )

addEntry(`int hash, K key, V value, int bucketIndex`)

的作用是**新增Entry。**将“key-value”插入指定位置，bucketIndex是位置索引。

(01) addEntry()一般用在 **新增Entry可能导致“HashMap的实际容量”超过“阈值”**

(02) createEntry() 一般用在 **新增Entry不会导致“HashMap的实际容量”超过“阈值”**

putAll(`Map<? extends K, ? extends V> m`)

将"m"的全部元素都添加到HashMap中。

remove(object key)

删除“键为key”元素。

遍历

遍历HashMap的键值对

第一步：根据entrySet()获取HashMap的“键值对”的Set集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。

entrySet()实际上是通过newEntryIterator()实现的。通过entrySet()获取到的Iterator的next()方法去遍历HashMap时，实际上调用的是 nextEntry() 。而nextEntry()的实现方式：先遍历Entry(根据Entry在table中的序号，从小到大的遍历)；然后对每个Entry(即每个单向链表)，逐个遍历。

遍历HashMap的键

第一步：根据keySet()获取HashMap的“键”的Set集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。

遍历HashMap的值

第一步：根据value()获取HashMap的“值”的集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。