hashMap 散式列表

有一个叫做table大小是16的Entry数组。

这个table数组存储了Entry类的对象。HashMap类有一个叫做Entry的内部类。这个Entry类包含了key-value作为实例变量。我们来看下Entry类的结构。Entry类的结构：

static class Entry implements Map.Entry

{

final K key;

V value;

Entry next; //链表结构的基础

final int hash;

...//More code goes here

} `

每当往hashmap里面存放key-value对的时候，都会为它们实例化一个Entry对象，这个Entry对象就会存储在前面提到的Entry数组table中。现在你一定很想知道，上面创建的Entry对象将会存放在具体哪个位置（在table中的精确位置）。答案就是，根据key的hashcode()方法计算出来的hash值（来决定）。hash值用来计算key在Entry数组的索引。

我们往hashmap放了4个key-value对，但是看上去好像只有2个元素！！！这是因为，如果两个元素有相同的hashcode，它们会被放在同一个索引上。问题出现了，该怎么放呢？原来它是以链表(LinkedList)的形式来存储的(逻辑上)。

Put()方法:

对key做null检查。如果key是null，会被存储到table[0]，因为null的hash值总是0。

key的hashcode()方法会被调用，然后计算hash值。hash值用来找到存储Entry对象的数组的索引。有时候hash函数可能写的很不好，所以JDK的设计者添加了另一个叫做hash()的方法，它接收刚才计算的hash值作为参数。

indexFor(hash,table.length)用来计算在table数组中存储Entry对象的精确的索引。

在我们的例子中已经看到，如果两个key有相同的hash值(也叫冲突)，他们会以链表的形式来存储。所以，这里我们就迭代链表。

* 如果在刚才计算出来的索引位置没有元素，直接把Entry对象放在那个索引上。
* 如果索引上有元素，然后会进行迭代，一直到Entry->next是null。当前的Entry对象变成链表的下一个节点。
* 如果我们再次放入同样的key会怎样呢？逻辑上，它应该替换老的value。事实上，它确实是这么做的。在迭代的过程中，会调用equals()方法来检查key的相等性(key.equals(k))，如果这个方法返回true，它就会用当前Entry的value来替换之前的value。

****Get:****

对key进行null检查。如果key是null，table[0]这个位置的元素将被返回。

key的hashcode()方法被调用，然后计算hash值。

indexFor(hash,table.length)用来计算要获取的Entry对象在table数组中的精确的位置，使用刚才计算的hash值。

在获取了table数组的索引之后，会迭代链表，调用equals()方法检查key的相等性，如果equals()方法返回true，get方法返回Entry对象的value，否则，返回null。

****要牢记以下关键点：****

* HashMap有一个叫做Entry的内部类，它用来存储key-value对。
* 上面的Entry对象是存储在一个叫做table的Entry数组中。
* table的索引在逻辑上叫做“桶”(bucket)，它存储了链表的第一个元素。
* key的hashcode()方法用来找到Entry对象所在的桶。
* 如果两个key有相同的hash值，他们会被放在table数组的同一个桶里面。
* key的equals()方法用来确保key的唯一性。
* value对象的equals()和hashcode()方法根本一点用也没有。

LinkedList 是一个继承于AbstractSequentialList的双向链表。它也可以被当作堆栈、队列或双端队列进行操作。

LinkedList 实现 List 接口，能对它进行队列操作。  
LinkedList 实现 Deque 接口，即能将LinkedList当作双端队列使用。  
LinkedList 实现了Cloneable接口，即覆盖了函数clone()，能克隆。  
LinkedList 实现java.io.Serializable接口，这意味着LinkedList支持序列化，能通过序列化去传输。  
LinkedList 是非同步的。

LinkedList底层的数据结构是基于双向循环链表的

并发编程实践中，ConcurrentHashMap是一个经常被使用的数据结构,大量的利用了volatile，final，CAS等lock-free技术来减少锁竞争对于性能的影响，无论对于Java并发编程的学习还是Java内存模型的理解，ConcurrentHashMap的设计以及源码都值得非常仔细的阅读与揣摩。

1. Iterator iter = list.iterator();
2. //2、通过循环迭代
3. //hasNext():判断是否存在下一个元素
4. **while**(iter.hasNext()){
5. //如果存在，则调用next实现迭代
6. //Object-->Integer-->int
7. **int** j=(**int**)iter.next();  //把Object型强转成int型
8. System.out.println(j);
9. }

hashTable的key不能为空 ,线程安全

Hashmapkey可以为空,线程不安全