Java中对HashMap的深度分析与比较 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022_Java_E4_B8_AD_E5_AF_B9_c104_252442.htm 在Java的世界里，无论类还是各种数据，其结构的处理是整个程序的逻辑以及性能的关键。由于本人接触了一个有关性能与逻辑同时并存的问题，于是就开始研究这方面的问题。找遍了大大小小的论坛，也把《Java 虚拟机规范》，《apress,.java.collections.(2001),.bm.ocr.6.0.shareconnector》，和《Thinking in Java》翻了也找不到很好的答案，于是一气之下把JDK的 src 解压出来研究，扩然开朗，遂写此文，跟大家分享感受和顺便验证我理解还有没有漏洞。 这里就拿HashMap来研究吧。 HashMap可谓JDK的一大实用工具，把各个Object映射起来，实现了“ 键－－值 ”对应的快速存取。但实际里面做了些什么呢？ 在这之前，先介绍一下负载因子和容量的属性。大家都知道其实一个 HashMap 的实际容量就 因子*容量，其默认值是 16×0.75＝12；这个很重要，对效率很一定影响！当存入HashMap的对象超过这个容量时，HashMap 就会重新构造存取表。这就是一个大问题，我后面慢慢介绍，反正，如果你已经知道你大概要存放多少个对象，最好设为该实际容量的能接受的数字。 两个关键的方法，put和get： 先有这样一个概念，HashMap是声明了 Map，Cloneable, Serializable 接口，和继承了 AbstractMap 类，里面的 Iterator 其实主要都是其内部类HashIterator 和其他几个 iterator 类实现，当然还有一个很重要的继承了Map.Entry 的 Entry 内部类，由于大家都有源代码，大家有兴趣可以看看这部分，我主要想

说明的是 Entry 内部类。它包含了hash，value，key 和next 这四个属性，很重要。put的源码如下public Object put(Object key, Object value) {Object k = maskNull(key). 这个就是判断键值是否为空，并不很深奥，其实如果为空，它会返回一个static Object 作为键值，这就是为什么HashMap允许空键值的原因。int hash = hash(k).int i = indexFor(hash, table.length). 这连续的两步就是 HashMap 最牛的地方！研究完我都汗颜了，其中 hash 就是通过 key 这个Object的 hashcode 进行 hash，然后通过 indexFor 获得在Object table的索引值。 table？？？不要惊讶，其实HashMap也神不到哪里去，它就是用 table 来放的。最牛的就是用 hash 能正确的返回索引。其中的hash算法，我跟JDK的作者 Doug 联系过，他建议我看看《The art of programing vol3》可恨的是，我之前就一直在找，我都找不到，他这样一提，我就更加急了，可惜口袋空空啊！！！ 不知道大家有没有留意 put 其实是一个有返回的方法，它会把相同键值的 put 覆盖掉并返回旧的值！如下方法彻底说明了 HashMap 的结构，其实就是一个表加上在相应位置的Entry的链表：for (Entry e = table[i]. e != null. e = e.next) { if (e.hash == hash amp. eq(k, e.key)) { Object oldvalue = e.value. e.value = value. //把新的值赋予给对应键值。 e.recordACCESS(this). //空方法，留待实现 return oldvalue. //返回相同键值的对应的旧的值。}}modCount . //结构性更改的次数addEntry(hash, k, value, i). //添加新元素， 关键所在！return null. //没有相同的键值返回}我们把关键的方法拿出来分析：void addEntry(int hash, Object key, Object value, int bucketIndex) {table[bucketIndex] = new Entry(hash, key, value, table[bucketIndex]). 因为 hash 的算法有可

能令不同的键值有相同的hash码并有相同的table索引，如：key＝“ 33 ”和key＝Object g的hash都是－8901334，那它经过indexfor之后的索引一定都为i，这样在new的时候这个Entry的next就会指向这个原本的table[i]，再有下一个也如此，形成一个链表，和put的循环对定e.next获得旧的值。到这里，HashMap的结构，大家也十分明白了吧？if (size >= threshold) //这个threshold就是能实际容纳的量resize(2 * table.length). //超出这个容量就会将Object table重构 所谓的重构也不神，就是建一个两倍大的table（我在别的论坛上看到有人说是两倍加1，把我骗了），然后再一个个indexfor进去！注意！！这就是效率！！如果你能让你的HashMap不需要重构那么多次，效率会大大提高！ 说到这里也差不多了，get比put简单得多，大家，了解put，get也差不了多少了。对于collections我是认为，它是适合广泛的，当不完全适合特有的，如果大家的程序需要特殊的用途，自己写吧，其实很简单。（作者是这样跟我说的，他还建议我用LinkedHashMap，我看了源码以后发现，LinkHashMap其实就是继承HashMap的，然后override相应的方法，有兴趣的同人，自己looklook）建个 Object table，写相应的算法，就ok啦。 举个例子吧，像Vector，list 啊什么的其实都很简单，最多就多了的同步的声明，其实如果要实现像Vector那种，插入，删除不多的，可以用一个Object table来实现，按索引存取，添加等。 如果插入，删除比较多的，可以建两个Object table，然后每个元素用含有next结构的，一个table存，如果要插入到i，但是i已经有元素，用next连起来，然后size＋＋，并在另一个table记录其位置。 100Test 下载频道开通，各类考试题目直接下载。详