



9

举报

HashMap、HashSet和HashTable详解

原创 bear_wr 最后发布于2016-08-22 15:11:58 阅读数 7632 ☆ 收藏

尾

HashMap 和 HashSet 是 Java Collection Framework 的两个重要成员，其中 HashMap 是 Map 接口的常用实现类，HashSet 是 Set 接口的常用实现类。虽然 HashMap 和 HashSet 实现的接口规范不同，但它们底层的 Hash 存储机制完全一样，甚至 HashSet 本身就采用 HashMap 来实现的。

通过 HashMap、HashSet 的源代码分析其 Hash 存储机制

实际上，HashSet 和 HashMap 之间有很多相似之处，对于 HashSet 而言，系统采用 Hash 算法决定集合元素的存储位置，这样可以保证能快速存、取集合元素；对于 HashMap 而言，系统 key-value 当成一个整体进行处理，系统总是根据 Hash 算法来计算 key-value 的存储位置，这样可以保证能快速存、取 Map 的 key-value 对。

在介绍集合存储之前需要指出一点：虽然集合号称存储的是 Java 对象，但实际上并不会真正将 Java 对象放入 Set 集合中，只是在 Set 集合中保留这些对象的引用。也就是说：Java 集合实际上是多个引用变量所组成的集合，这些引用变量指向实际的 Java 对象。

集合和引用

就像引用类型的数组一样，当我们把 Java 对象放入数组之时，并不是真正的把 Java 对象放入数组中，只是把对象的引用放入数组中，每个数组元素都是一个引用变量。

HashMap

当程序试图将多个 key-value 放入 HashMap 中时，以如下代码片段为例：

```
1 HashMap<String , Double> map = new HashMap<String , Double>();
2 map.put("语文" , 80.0);
3 map.put("数学" , 89.0);
4 map.put("英语" , 78.2);
```

HashMap 采用一种所谓的“Hash 算法”来决定每个元素的存储位置。

当程序执行 map.put(“语文”, 80.0); 时，系统将调用“语文”的 hashCode() 方法得到其 hashCode 值——每个 Java 对象都有 hashCode() 方法，都可通过该方法获取它的 hashCode 值。得到这个对象的 hashCode 值之后，系统会根据该 hashCode 值来决定该元素的存储位置。

我们可以看 HashMap 类的 put(K key, V value) 方法的源代码：

```
1 public V put(K key, V value)
2 {
3     // 如果 key 为 null, 调用 putForNullKey 方法进行处理
4     if (key == null)
5         return putForNullKey(value);
6     // 根据 key 的 keyCode 计算 Hash 值
7     int hash = hash(key.hashCode());
8     // 搜索指定 hash 值在对应 table 中的索引
9     int i = indexFor(hash, table.length);
10    // 如果 i 索引处的 Entry 不为 null, 通过循环不断遍历 e 元素的下一个元素
11    for (Entry<K,V> e = table[i]; e != null; e = e.next)
12    {
13        Object k;
14        // 找到指定 key 与需要放入的 key 相等 (hash 值相同)
15        // 通过 equals 比较放回 true)
16        if (e.hash == hash && ((k = e.key) == key
17            || key.equals(k)))
18        {
19            V oldValue = e.value;
20            e.value = value;
```

```

21         e.recordAccess(this);
22         return oldValue;
23     }
24 }
25 // 如果 i 索引处的 Entry 为 null, 表明此处还没有 Entry
26 modCount++;
27 // 将 key、value 添加到 i 索引处
28 addEntry(hash, key, value, i);
29 return null;
30 }

```

上面程序中用到了一个重要的内部接口：Map.Entry，每个 Map.Entry 其实就是一个 key-value 对。从上面程序中可以看出：当系统决定存储 HashMap 中的 key-value 对时，完全没有考虑 Entry 中的 value，仅仅只是根据 key 来计算并决定每个 Entry 的存储位置。这也说明了前面的结论：我们完全可以把 Map 集合中的 value 当成 key 的附属，当系统决定了 key 的存储位置之后，value 随之保存在那里即可。

上面方法提供了一个根据 hashCode() 返回值来计算 Hash 码的方法：hash()，这个方法是一个纯粹的数学计算，其方法如下：

```

1 static int hash(int h)
2 {
3     h ^= (h >>> 20) ^ (h >>> 12);
4     return h ^ (h >>> 7) ^ (h >>> 4);
5 }

```

对于任意给定的对象，只要它的 hashCode() 返回值相同，那么程序调用 hash(int h) 方法所计算得到的 Hash 码值总是相同的。接下来程序会调用 indexFor(int h, length) 方法来计算该对象应该保存在 table 数组的哪个索引处。indexFor(int h, int length) 方法的代码如下：

```

1 static int indexFor(int h, int length)
2 {
3     return h & (length-1);
4 }

```

这个方法非常巧妙，它总是通过 $h \& (table.length - 1)$ 来得到该对象的保存位置——而 HashMap 底层数组的长度总是 2 的 n 次方，这一点可参看后面关于 HashM 构造器的介绍。

当 length 总是 2 的倍数时， $h \& (length - 1)$ 将是一个非常巧妙的设计：假设 $h=5, length=16$ ，那么 $h \& length - 1$ 将得到 5；如果 $h=6, length=16$ ，那么 $h \& length - 1$ 将得到 6 如果 $h=15, length=16$ ，那么 $h \& length - 1$ 将得到 15；但是当 $h=16$ 时， $length=16$ 时，那么 $h \& length - 1$ 将得到 0 了；当 $h=17$ 时， $length=16$ 时，那么 $h \& length - 1$ 将得到 1 了 这样保证计算得到的索引值总是位于 table 数组的索引之内。

根据上面 put 方法的源代码可以看出，当程序试图将一个 key-value 对放入 HashMap 中时，程序首先根据该 key 的 hashCode() 返回值决定该 Entry 的存储位置。如果两个 Entry 的 key 的 hashCode() 返回值相同，那它们的存储位置相同。如果这两个 Entry 的 key 通过 equals 比较返回 true，新添加 Entry 的 value 将覆盖集合中原有 Entry 的 value，但 key 不会覆盖。如果这两个 Entry 的 key 通过 equals 比较返回 false，新添加的 Entry 将与集合中原有 Entry 形成 Entry 链，而且新加的 Entry 位于 Entry 链的头部——具体说明继续看 addEntry() 方法的说明。

当向 HashMap 中添加 key-value 对，由其 key 的 hashCode() 返回值决定该 key-value 对（就是 Entry 对象）的存储位置。当两个 Entry 对象的 key 的 hashCode 返回值相同时，将由 key 通过 equals() 比较值决定是采用覆盖行为（返回 true），还是产生 Entry 链（返回 false）。

上面程序中还调用了 addEntry(hash, key, value, i); 代码，其中 addEntry 是 HashMap 提供的一个包访问权限的方法，该方法仅用于添加一个 key-value 对。下面该方法的代码：

```

1 void addEntry(int hash, K key, V value, int bucketIndex)
2 {
3     // 获取指定 bucketIndex 索引处的 Entry
4     Entry<K,V> e = table[bucketIndex]; // ①
5     // 将新创建的 Entry 放入 bucketIndex 索引处，并让新的 Entry 指向原来的 Entry
6     table[bucketIndex] = new Entry<K,V>(hash, key, value, e);
7     // 如果 Map 中的 key-value 对的数量超过了极限
8     if (size++ >= threshold)
9         // 把 table 对象的长度扩充到 2 倍。

```

```

10         resize(2 * table.length);    // ②
11     }

```

上面方法的代码很简单，但其中包含了一个非常优雅的设计：系统总是将新添加的 Entry 对象放入 table 数组的 bucketIndex 索引处——如果 bucketIndex 索引处已经有了一个 Entry 对象，那新添加的 Entry 对象指向原有的 Entry 对象（产生一个 Entry 链），如果 bucketIndex 索引处没有 Entry 对象，也就是上面程序①号代码的 e 变量是 null，也就是新放入的 Entry 对象指向 null，也就是没有产生 Entry 链。

根据上面代码可以看出，在同一个 bucket 存储 Entry 链的情况下，新放入的 Entry 总是位于 bucket 中，而最早放入该 bucket 中的 Entry 则位于这个 Entry 链的最末端。

上面程序中还有这样两个变量：

- 1 * size: 该变量保存了该 HashMap 中所包含的 key-value 对的数量。
- 2 * threshold: 该变量包含了 HashMap 能容纳的 key-value 对的极限，它的值等于 HashMap 的容量乘以负载因子 (load factor)。

从上面程序中②号代码可以看出，当 size++ >= threshold 时，HashMap 会自动调用 resize 方法扩充 HashMap 的容量。每扩充一次，HashMap 的容量就增大一倍。

上面程序中使用的 table 其实就是一个普通数组，每个数组都有一个固定的长度，这个数组的长度就是 HashMap 的容量。HashMap 包含如下几个构造器：

- 1 * HashMap(): 构建一个初始容量为 16，负载因子为 0.75 的 HashMap。
- 2 * HashMap(int initialCapacity): 构建一个初始容量为 initialCapacity，负载因子为 0.75 的 HashMap。
- 3 * HashMap(int initialCapacity, float loadFactor): 以指定初始容量、指定的负载因子创建一个 HashMap。

当创建一个 HashMap 时，系统会自动创建一个 table 数组来保存 HashMap 中的 Entry，下面是 HashMap 中一个构造器的代码：

```

1  // 以指定初始化容量、负载因子创建 HashMap
2  public HashMap(int initialCapacity, float loadFactor)
3  {
4      // 初始容量不能为负数
5      if (initialCapacity < 0)
6          throw new IllegalArgumentException(
7              "Illegal initial capacity: " +
8              initialCapacity);
9      // 如果初始容量大于最大容量，让出容量
10     if (initialCapacity > MAXIMUM_CAPACITY)
11         initialCapacity = MAXIMUM_CAPACITY;
12     // 负载因子必须大于 0 的数值
13     if (loadFactor <= 0 || Float.isNaN(loadFactor))
14         throw new IllegalArgumentException(
15             loadFactor);
16     // 计算出大于 initialCapacity 的最小的 2 的 n 次方值。
17     int capacity = 1;
18     while (capacity < initialCapacity)
19         capacity <<= 1;
20     this.loadFactor = loadFactor;
21     // 设置容量极限等于容量 * 负载因子
22     threshold = (int)(capacity * loadFactor);
23     // 初始化 table 数组
24     table = new Entry[capacity];    // ①
25     init();
26 }

```

上面代码中粗体字代码包含了一个简洁的代码实现：找出大于 initialCapacity 的、最小的 2 的 n 次方值，并将其作为 HashMap 的实际容量（由 capacity 变量保存）。例如给定 initialCapacity 为 10，那么该 HashMap 的实际容量就是 16。

程序①号代码处可以看到：table 的实质就是一个数组，一个长度为 capacity 的数组。

对于 HashMap 及其子类而言，它们采用 Hash 算法来决定集合中元素的存储位置。当系统开始初始化 HashMap 时，系统会创建一个长度为 capacity 的 Entry 数

组，这个数组里可以存储元素的位置被称为“桶（bucket）”，每个 bucket 都有其指定索引，系统可以根据其索引快速访问该 bucket 里存储的元素。

无论何时，HashMap 的每个“桶”只存储一个元素（也就是一个 Entry），由于 Entry 对象可以包含一个引用变量（就是 Entry 构造器的的最后一个参数）用于指向一个 Entry，因此可能出现的情况是：HashMap 的 bucket 中只有一个 Entry，但这个 Entry 指向另一个 Entry ——这就形成了一个 Entry 链。如图 所示：

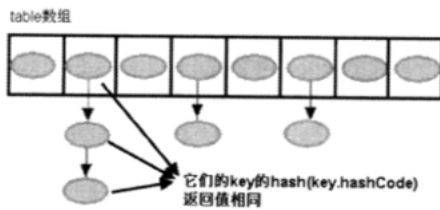


图 3.8 HashMap 的存储

HashMap 的读取实现

当 HashMap 的每个 bucket 里存储的 Entry 只是单个 Entry ——也就是没有通过指针产生 Entry 链时，此时的 HashMap 具有最好的性能：当程序通过 key 取出 value 时，系统只要先计算出该 key 的 hashCode() 返回值，在根据该 hashCode 返回值找出该 key 在 table 数组中的索引，然后取出该索引处的 Entry，最后返回该 key 对应的 value 即可。看 HashMap 类的 get(K key) 方法代码：

```

1  public V get(Object key)
2  {
3      // 如果 key 是 null，调用 getForNullKey 取出对应的 value
4      if (key == null)
5          return getForNullKey();
6      // 根据该 key 的 hashCode 值计算它的 hash 码
7      int hash = hash(key.hashCode());
8      // 直接取出 table 数组中指定索引处的值，
9      for (Entry<K,V> e = table[indexFor(hash, table.length)];
10         e != null;
11         // 搜索该 Entry 链的下一个 Entry
12         e = e.next)           // ①
13     {
14         Object k;
15         // 如果该 Entry 的 key 与被搜索 key 相同
16         if (e.hash == hash && ((k = e.key) == key
17             || key.equals(k)))
18             return e.value;
19     }
20     return null;
21 }
```

从上面代码中可以看出，如果 HashMap 的每个 bucket 里只有一个 Entry 时，HashMap 可以根据索引、快速地取出该 bucket 里的 Entry；在发生“Hash 冲突”的情况下，单个 bucket 里存储的不是一个 Entry，而是一个 Entry 链，系统只能必须按顺序遍历每个 Entry，直到找到想搜索的 Entry 为止——如果恰好要搜索的 Entry 位于该 Entry 链的最末端（该 Entry 是最早放入该 bucket 中），那系统必须循环到最后才能找到该元素。

归纳起来简单地说，HashMap 在底层将 key-value 当成一个整体进行处理，这个整体就是一个 Entry 对象。HashMap 底层采用一个 Entry[] 数组来保存所有的 key-value 对，当需要存储一个 Entry 对象时，会根据 Hash 算法来决定其存储位置；当需要取出一个 Entry 时，也会根据 Hash 算法找到其存储位置，直接取出该 Entry。由此可见：HashMap 之所以能快速存、取它所包含的 Entry，完全类似于现实生活中母亲从小教我们的：不同的东西要放在不同的位置，需要时才能快速到它。

当创建 HashMap 时，有一个默认的负载因子（load factor），其默认值为 0.75，这是时间和空间成本上一种折衷：增大负载因子可以减少 Hash 表（就是那个 Entry 数组）所占用的内存空间，但会增加查询数据的时间开销，而查询是最频繁的操作（HashMap 的 get() 与 put() 方法都要用到查询）；减小负载因子会提高数据查询的性能，但会增加 Hash 表所占用的内存空间。

掌握了上面知识之后，我们可以在创建 HashMap 时根据实际需要适当地调整 load factor 的值；如果程序比较关心空间开销、内存比较紧张，可以适当地增加负载因子；如果程序比较关心时间开销，内存比较宽裕则可以适当的减少负载因子。通常情况下，程序员无需改变负载因子的值。

如果开始就知道 HashMap 会保存多个 key-value 对，可以在创建时就使用较大的初始化容量，如果 HashMap 中 Entry 的数量一直不会超过极限容量（capacity * load factor），HashMap 就无需调用 resize() 方法重新分配 table 数组，从而保证较好的性能。当然，开始就将初始容量设置太高可能会浪费空间（系统需要创建一个长度为 capacity 的 Entry 数组），因此创建 HashMap 时初始化容量设置也需要小心对待。

HashSet

对于HashSet而言，它的底层是基于HashMap实现的。HashSet底层使用HashMap来保存所有元素。源码如下：

```
1 public class HashSet<E>
2     extends AbstractSet<E>
3     implements Set<E>, Cloneable, java.io.Serializable
4 {
5     static final long serialVersionUID = -5024744406713321676L;
6
7     private transient HashMap<E, Object> map;
8     private static final Object PRESENT = new Object();
9
10
11     public HashSet() {
12         map = new HashMap<>();
13     }
14
15     public HashSet(Collection<? extends E> c) {
16         map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
17         addAll(c);
18     }
19
20
21     public HashSet(int initialCapacity, float loadFactor) {
22         map = new HashMap<>(initialCapacity, loadFactor);
23     }
24
25
26     public HashSet(int initialCapacity) {
27         map = new HashMap<>(initialCapacity);
28     }
29
30
31     HashSet(int initialCapacity, float loadFactor, boolean dummy) {
32         map = new LinkedHashMap<>(initialCapacity, loadFactor);
33     }
34
35
36     public Iterator<E> iterator() {
37         return map.keySet().iterator();
38     }
39
40
41     public int size() {
42         return map.size();
43     }
44
45
46     public boolean isEmpty() {
47         return map.isEmpty();
48     }
49
50 }
```

```

51     public boolean contains(Object o) {
52         return map.containsKey(o);
53     }
54
55
56     public boolean add(E e) {
57         return map.put(e, PRESENT) == null;
58     }
59
60
61     public boolean remove(Object o) {
62         return map.remove(o) == PRESENT;
63     }
64
65
66     public void clear() {
67         map.clear();
68     }
69
70     ...

```

HashSet的实现其实非常简单，它只是封装了一个HashMap对象来存储所有的集合元素。所有放入HashSet中的集合元素实际上由HashMap的key来保存，而HashMap的value则存储了一个PRESENT，它是一个静态的Object对象。

HashSet的绝大部分方法都是通过调用HashMap的方法来实现的，因此HashSet和HashMap两个集合在实现本质上是相同的。

注意：由于HashSet的add()方法添加集合元素实际上转变为调用HashMap的put()方法来添加key-value对，当新放入HashMap的Entry中key与集合中原有Entry的key相同（hashCode()返回值相等，通过equals比较也返回true）时，新添加的Entry的value将覆盖原来Entry的value，但key不会有任何改变。因此，如果向HashSet添加一个已经存在的元素，新添加的集合元素（底层由HashMap的key保存）不会覆盖已有的集合元素。

HashTable与HashMap区别

首先看看两个的继承关系

```

1  public class Hashtable<K,V>
2      extends Dictionary<K,V>
3      implements Map<K,V>, Cloneable, java.io.Serializable
4
5  public class HashMap<K,V> extends AbstractMap<K,V>
6      implements Map<K,V>, Cloneable, Serializable

```

HashTable如何添加数据的呢？看看他的put方法：

```

1  public synchronized V put(K key, V value) {
2      // Make sure the value is not null
3      if (value == null) {
4          throw new NullPointerException();
5      }
6
7      // Makes sure the key is not already in the hashtable.
8      Entry<?,?> tab[] = table;
9      int hash = key.hashCode();
10     int index = (hash & 0x7FFFFFFF) % tab.length;
11     @SuppressWarnings("unchecked")
12     Entry<K,V> entry = (Entry<K,V>)tab[index];
13     for(;; entry != null ; entry = entry.next) {
14         if ((entry.hash == hash) && entry.key.equals(key)) {
15             V old = entry.value;
16             entry.value = value;
17             return old;
18         }

```

```

19     }
20
21     addEntry(hash, key, value, index);
22     return null;
23 }
24
25
26 private void addEntry(int hash, K key, V value, int index) {
27     modCount++;
28
29     Entry<?,?> tab[] = table;
30     if (count >= threshold) {
31         // Rehash the table if the threshold is exceeded
32         rehash();
33
34         tab = table;
35         hash = key.hashCode();
36         index = (hash & 0x7FFFFFFF) % tab.length;
37     }
38
39     // Creates the new entry.
40     @SuppressWarnings("unchecked")
41     Entry<K,V> e = (Entry<K,V>) tab[index];
42     tab[index] = new Entry<>(hash, key, value, e);
43     count++;
44 }

```

Hashtable 中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。

Hashtable中，key和value都不允许出现null值。

在HashMap中，null可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为null。当get()方法返回null值时，即可以表示HashMap中没有该键，也可以用表示该键所对应的值为null。因此，在HashMap中不能由get()方法来判断HashMap中是否存在某个键，而应该用containsKey()方法来判断。

```

1  protected void rehash() {
2      int oldCapacity = table.length;
3      Entry<?,?>[] oldMap = table;
4
5      // overflow-conscious code
6      int newCapacity = (oldCapacity << 1) + 1;
7      if (newCapacity - MAX_ARRAY_SIZE > 0) {
8          if (oldCapacity == MAX_ARRAY_SIZE)
9              // Keep running with MAX_ARRAY_SIZE buckets
10             return;
11         newCapacity = MAX_ARRAY_SIZE;
12     }
13     Entry<?,?>[] newMap = new Entry<?,?>[newCapacity];
14
15     modCount++;
16     threshold = (int)Math.min(newCapacity * loadFactor, MAX_ARRAY_SIZE + 1);
17     table = newMap;
18
19     for (int i = oldCapacity ; i-- > 0 ; ) {
20         for (Entry<K,V> old = (Entry<K,V>)oldMap[i] ; old != null ; ) {
21             Entry<K,V> e = old;
22             old = old.next;
23
24             int index = (e.hash & 0x7FFFFFFF) % newCapacity;
25             e.next = (Entry<K,V>)newMap[index];
26             newMap[index] = e;
27         }
28     }

```



```
29    }
```

Hashtable和HashMap它们两个内部实现方式的数组的初始大小和扩容的方式。**HashTable**中**hash**数组默认大小是**11**，增加的方式是 **old*2+1**。**HashMap**中**hash**数组的默认大小是**16**，而且一定是**2**的指数。

另外，两个遍历方式的内部实现上不同。

Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式。

哈希值的使用不同，HashTable直接使用对象的hashCode。而HashMap重新计算hash值。

下面整理下一些关于HashMap的面试题：

引用于此：<http://www.importnew.com/7099.html>

“你知道HashMap的工作原理吗？” “你知道HashMap的get()方法的工作原理吗？”

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()方法传递键和值时，我们对键调用hashCode()方法，返回的hashCode用于找到bucket位置来储存Entry对象。”这里关键点在于指出，HashMap是在bucket中储存键对象和值对象，作为Map.Entry。这一点有助于理解获取对象的逻辑。

HashMap中的碰撞探测(collision detection)以及碰撞的解决方法：

“当两个对象的hashcode相同会发生什么？” 因为hashcode相同，所以它们的bucket位置相同，‘碰撞’会发生。因为HashMap使用链表存储对象，这个Entry(包含有值对的Map.Entry对象)会存储在链表中。”这个答案非常的合理，虽然有很多种处理碰撞的方法，这种方法是最简单的，也正是HashMap的处理方法。

“如果两个键的hashcode相同，你如何获取值对象？”

当我们调用get()方法，HashMap会使用键对象的hashcode找到bucket位置，找到bucket位置之后，会调用keys.equals()方法去找到链表中正确的节点，最终找到找的值对象。

许多情况下，面试者会在这个环节中出错，因为他们混淆了hashCode()和equals()方法。因为在此之前hashCode()屡屡出现，而equals()方法仅仅在获取值对象的时候才出现。一些优秀的开发者会指出使用不可变的、声明作final的对象，并且采用合适的equals()和hashCode()方法的话，将会减少碰撞的发生，提高效率。不可性使得能够缓存不同键的hashcode，这将提高整个获取对象的速度，使用String，Integer这样的wrapper类作为键是非常好的选择。

“如果HashMap的大小超过了负载因子(load factor)定义的容量，怎么办？”

默认的负载因子大小为0.75，也就是说，当一个map填满了75%的bucket时候，和其它集合类(如ArrayList等)一样，将会创建原来HashMap大小的两倍的bucket数组，来重新调整map的大小，并将原来的对象放入新的bucket数组中。这个过程叫作rehashing，因为它调用hash方法找到新的bucket位置。

“你了解重新调整HashMap大小存在什么问题吗？”

当重新调整HashMap大小的时候，确实存在条件竞争，因为如果两个线程都发现HashMap需要重新调整大小了，它们会同时试着调整大小。在调整大小的过程中存储在链表中的元素的次序会反过来，因为移动到新的bucket位置的时候，HashMap并不会将元素放在链表的尾部，而是放在头部，这是为了避免尾部遍历(tail traversing)。如果条件竞争发生了，那么就死循环了。

为什么String, Integer这样的wrapper类适合作为键？

String, Integer这样的wrapper类作为HashMap的键是再适合不过了，而且String最为常用。因为String是不可变的，也是final的，而且已经重写了equals()和hashCode()方法了。其他的wrapper类也有这个特点。不可变性是必要的，因为为了要计算hashCode()，就要防止键值改变，如果键值在放入时和获取时返回不同的hashcode的话，那么就不能从HashMap中找到你想要的对象。不可变性还有其他的优点如线程安全。如果你可以仅仅通过将某个field声明成final就能保证hashCode是不变的，那么请这么做吧。因为获取对象的时候要用到equals()和hashCode()方法，那么键对象正确的重写这两个方法是非常重要的。如果两个不相的对象返回不同的hashcode的话，那么碰撞的几率就会小些，这样就能提高HashMap的性能。

我们可以使用自定义的对象作为键吗？

当然你可能使用任何对象作为键，只要它遵守了equals()和hashCode()方法的定义规则，并且当对象插入到Map中之后将不会再改变了。如果这个自定义对象是不变的，那么它已经满足了作为键的条件，因为当它创建之后就已经不能改变了。

我们可以使用CocurrentHashMap来代替Hashtable吗？

这是另外一个很热门的面试题，因为ConcurrentHashMap越来越多人用了。我们知道Hashtable是synchronized的，但是ConcurrentHashMap同步性能更好，因为仅仅根据同步级别对map的一部分进行上锁。ConcurrentHashMap当然可以代替HashTable，但是HashTable提供更强的线程安全性。看看[这篇博客](#)查看Hashtable和ConcurrentHashMap的区别。

总结

HashMap的工作原理

HashMap基于hashing原理，我们通过put()和get()方法储存和获取对象。当我们将键值对传递给put()方法时，它调用键对象的hashCode()方法来计算hashcode，后找到bucket位置来储存值对象。当获取对象时，通过键对象的equals()方法找到正确的键值对，然后返回值对象。HashMap使用链表来解决碰撞问题，当发生碰了，对象将会储存在链表的下一个节点中。HashMap在每个链表节点中储存键值对对象。

当两个不同的键对象的hashcode相同时会发生什么？ 它们会储存在同一个bucket位置的链表中。键对象的equals()方法用来找到键值对。

参考：《疯狂java讲义》

<http://www.cnblogs.com/devinzhang/archive/2012/01/13/2321481.html>

<http://www.importnew.com/7099.html>

点赞

9

收藏

分享

...



bear_wr

发布了22 篇原创文章 · 获赞 80 · 访问量 26万+

私信

关注

HashMap 与HashTable的区别

阅读数 11万+

HashMap与HashTable的区别HashMap与Hashtable的区别是面试中经常遇到的一个问题。这个问题看... 博文 来自： wangxing233...



想对作者说点什么

从入门到精通，Java学习路线导航（附学习资源）

阅读数 7万+

引言最近也有很多人来向我"请教"，他们大都是一些刚入门的新手，还不了解这个行业，也不知道从何... 博文 来自： java_sha的博客

相见恨晚的超实用网站

阅读数 8万+

相见恨晚的超实用网站持续更新中。。 博文 来自： 藏冰的博客

HashMap,HashTable,HashSet之间的区别

阅读数 1430

HashMap,HashTable,HashSet之间的区别1.实现接口的不同 3.执行效率2.线程安全性，同步 4.key,valu... 博文 来自： 像风一样奔跑



加拿大必去的10个旅游圣地

自助游加拿大

热门

广告

Java基础面试题（一）集合，hashmap，hashtable，hashset

阅读数 683

问题一：集合类，hashmap的具体实现方法，用法，hashmap，hashset，hashtable的区别。（转http... 博文 来自： 足球界的码农L...

花了20分钟，给女朋友们写了一个web版群聊程序

阅读数 23万+

参考博客[1]https://www.byteslounge.com/tutorials/java-ee-html5-websocket-example 博文

c++制作的植物大战僵尸，开源，一代二代结合游戏

阅读数 1万+

此游戏全部由本人自己制作完成。游戏大部分的素材来源于原版游戏素材，少部分搜集于网络，以及自... 博文 来自： 尔灵尔亿的博客

HashMap和Hashtable理解与对比

阅读数 1万+

一、概述HashMap和Hashtable的区别在面试的时候经常会被问到，那么它们有什么区别呢？这里谈一... 博文 来自： 随性的博客

源码阅读(19)：Java中主要的Map结构——HashMap容器（下1）

阅读数 5185

HashMap容器从字面的理解就是，基于Hash算法构造的Map容器。从数据结构的知识体系来说，Hash... 博文 来自： JAVA入门中

为何C++还没有被时代淘汰？

现在Python、Java地位上下不一，C++和他们相比，还能有什么价值？

广告 关闭

比特币原理**详解**

阅读量 15万+

一、什么是比特币比特币是一种电子货币，是一种基于密码学的货币，在2008年11月1日由中本聪发表... 博文 来自： zcg_74145489...

开源一个功能完整的SpringBoot项目框架

阅读量 5万+

福利来了，给大家带来一个福利。最近想了解一下有关Spring Boot的开源项目，看了很多开源的框架... 博文



hahahaha233

13篇文章

关注 排名:千里之外



九亿少女的梦@

26篇文章

关注 排名:千里之外



藏冰

130篇文章

关注 排名:千里之外



奔跑着的国风

265篇文章

关注 排名:4000+

HashMap和HashSet的区别

阅读量 7320

1、为什么用HashMap? HashMap是一个散列桶（数组和链表），它存储的内容是键值对(key-value)映... 博文 来自： 男人放得下自...

源码分析HashMap、Hashtable、HashSet的区别

阅读量 77

HashMap源码分析-基于JDK1.8基本结构1)、初始变量public class HashMap<K, V> ext... 博文 来自： MarinaTsang...

HashMap、HashSet、Hashtable的区别

阅读量 14

突然发现整理了很多笔记，应该放这里做备用Hashtable和HashMap主要区别：线程安全性，同步(sync... 博文 来自： weixin_34217...

那些拿到 60K Offer 的 AI 程序员，后来都怎么样了？

刚刚拿到阿里offer，工作地点杭州。值得去吗？

广告 关闭

Java学习的正确打开方式

阅读量 13万+

在博主认为，对于入门级学习java的最佳学习方法莫过于视频+博客+书籍+总结，前三者博主将淋漓尽... 博文 来自： 程序员宜春的...

Python——画一棵漂亮的樱花树（不同种樱花+玫瑰+圣诞树喔）

阅读量 16万+

最近翻到一篇知乎，上面有不少用Python（大多是turtle库）绘制的树图，感觉很漂亮，我整理了一下... 博文 来自： 碎片

程序员必须掌握的核心算法有哪些？

阅读量 22万+

由于我之前一直强调数据结构以及算法学习的重要性，所以就有一些读者经常问我，数据结构与算法应... 博文 来自： 帅地

网页实现一个简单的音乐播放器（大佬别看。(๑_๑)）

阅读量 5万+

今天闲着无事，就想写点东西。然后听了下歌，就打算写个播放器。于是乎用h5 audio的加上js简单的... 博文 来自： qq_44210563...

linux系列之常用运维命令整理笔录

阅读量 18万+

本博客记录工作中需要的linux运维命令，大学时候开始接触linux，会一些基本操作，可是都没有整理起... 博文 来自： Nicky's blog

反转！“只问了1个框架，就给了35K的Python岗”

学Python的程序员建议收藏！

学院广告 关闭

深度学习图像算法在内容安全领域的应用

阅读量 2940

互联网给人们生活带来便利的同时也隐含了大量不良信息，防范互联网平台有害内容传播引起了多方面... 博文 来自： LiveVideoStack

一生必看的纪录片

阅读量 1万+

下面按对自己的影响/感悟程度来排序《人生七年》概要：人生七年》又称作《56up》也是非常多的网... 博文 来自： 我的E家

2019年11月全国程序员工资统计，区块链工程师比算法工资高。

阅读数 2万+

我每个月第一天（也许是第二天，第三天），会爬招聘网站，并在CSDN发布。趋势本月全国程序员平...

博文 来自: [juwikuang的专栏](#)

一文读懂一台计算机是如何把数据发送给另一台计算机的

阅读数 6645

来源：苦逼的码农（ID：di201805）前言天各一方的两台计算机是如何通信的呢？在成千上万的计算机...

博文 来自: [Java团长的博客](#)

redis——相关问题汇总

阅读数 1万+

什么是redis？Redis 本质上是一个 Key-Value 类型的内存数据库，整个数据库加载在内存当中进行操...

博文 来自: [hebtu666](#)

自我心理治疗焦虑的方法

6.2万阅读



HashMap与HashTable的区别、HashMap与HashSet的关系

阅读数 1万+

HashTable的应用非常广泛，HashMap是新框架中用来代替HashTable的类，也就是说建议使用HashM...

博文 来自: [wl_idy的专栏](#)

Java中List集合介绍（炒鸡详细哟）

阅读数 7002

Java中List集合介绍（炒鸡详细哟）1，Java集合介绍作为一个程序猿，Java集合类可以说是我们在工...

博文 来自: [偏偏爱吃梨的...](#)

大学四年自学走来，这些私藏的实用工具/学习网站我贡献出来了

阅读数 27万+

大学四年，看课本是不可能一直看课本的了，对于学习，特别是自学，善于搜索网上的一些资源来辅助...

博文 来自: [帅地](#)

经典算法（5）杨辉三角

阅读数 6万+

杨辉三角 是经典算法，这篇博客对它的算法思想进行了讲解，并有完整的代码实现。...

博文 来自: [扬帆向海的博客](#)

C#中HashTable使用

阅读数 1112

添加元素Hashtable ht =new Hashtable();ht.Add(key,value);// key,value可以是任何类型删除元素ht.Re...

博文 来自: [syy1292的博客](#)

为啥国人偏爱Mybatis，而老外喜欢Hibernate/JPA呢？

阅读数 5万+

关于SQL和ORM的争论，永远都不会终止，我也一直在思考这个问题。昨天又跟群里的小伙伴进行了...

博文 来自: [十步杀一人-千...](#)

五款高效率黑科技神器工具，炸裂好用，省时间

阅读数 3万+

loonggg读完需要4分钟速读仅需2分钟感觉我好久好久没有给大家分享高质量的软件和插件了。今天周...

博文 来自: [非著名程序员](#)

通俗易懂地给女朋友讲：线程池的内部原理

阅读数 8万+

餐盘在灯光的照耀下格外晶莹剔透，女朋友拿起红酒杯轻轻地抿了一小口，对我说：“经常听你说线程...

博文 来自: [万猫学社](#)

中国麻将：世界上最早的区块链项目

阅读数 10万+

中国麻将：世界上最早的区块链项目最近区块链这个玩意又被市场搞的很是火热，相信大部分人都不太...

博文 来自: [gao_chun](#)

碎片化的时代，如何学习

阅读数 2万+

今天周末，和大家聊聊学习这件事情。在如今这个社会，我们的时间被各类 APP 撕的粉碎。刷知乎、...

博文 来自: [极客挖掘机](#)

加拿大必去的10个旅游圣地

加拿大旅游



项目中的if else太多了，该怎么重构？

阅读数 12万+

介绍最近跟着公司的大佬开发了一款IM系统，类似QQ和微信哈，就是聊天软件。我们有一部分业务逻...

博文

Python 编程开发 实用经验和技巧

阅读数 1万+

Python是一门很灵活的语言，也有很多实用的方法，有时候实现一个功能可以用多种方法实现，我这里...

博文 来自: [CUFEECR的...](#)

了解这些后，再去决定要不要买mac

阅读数 1671

当时买mac的初衷，只是想要个固态硬盘的笔记本，用来运行一些复杂的扑克软件。而看了当时所有的... 博文 来自: Diana5253的...

程序员求助：腾讯面试题，64匹马8个跑道，多少轮选出最快的四匹

阅读数 1万+

昨天，有网友私信我，说去阿里面试，彻底的被打击到了。问了为什么网上大量使用ThreadLocal的源... 博文 来自: ZYDX1898400...

【工具】（七）：win10常用工具整理!!!

阅读数 1万+

如题，本文主要为博主对电脑上安装的一些软件，所做的整理，当做备份用吧。一、分类系统工具办公... 博文 来自: Alfred的博客

[推动全社会公益氛围形成，使公益与空气和阳光一样触手可及。](#)
公益缺你不可，众多公益项目等你PICK——百度公益 让公益像「空气和阳光」一样触手可及！
[gongyi.baidu.com](#)

java中的HashTable,HashMap和HashSet的区别

阅读数 528

上篇博客中我们详细的分析了java集合《java中Map,List与Set的区别》。同时我们也对HashSet和Hash... 博文 来自: ZCC的专栏

爬虫小程序 - 爬取王者荣耀全皮肤

阅读数 12万+

王者荣耀全皮肤图片爬取 博文

字节跳动视频编解码面经

阅读数 6万+

三四月份投了字节跳动的实习（图形图像岗位），然后hr打电话过来问了一下会不会opengl，c++，sha... 博文

python学习方法总结(内附python全套学习资料)

阅读数 2万+

不要再问我python好不好学了 我之前做过半年少儿编程老师，一个小学四年级的小孩子都能在我的教... 博文

Python 基础（一）：入门必备知识

阅读数 9万+

Python 入门必备知识，你都掌握了吗？ 博文

 免税车 留学生购车 免税车

兼职程序员一般可以从什么平台接私活？

阅读数 16万+

这个问题我进行了系统性的总结，以下将进行言简意赅的说明和渠道提供，希望对各位小猿/小媛们有... 博文

Python十大装B语法

阅读数 24万+

Python 是一种代表简单思想的语言，其语法相对简单，很容易上手。不过，如果就此小视 Python 语法... 博文

数据库优化 - SQL优化

阅读数 14万+

以实际SQL入手，带你一步一步走上SQL优化之路！ 博文

2019年11月中国大陆编程语言排行榜


阅读数 5万+

2019年11月2日，我统计了某招聘网站，获得有效程序员招聘数据9万条。针对招聘信息，提取编程语... 博文

C++知识点 —— 整合（持续更新中）

阅读数 1万+

本文记录自己在自学C++过程中不同于C的一些知识点，适合于有C语言基础的同学阅读。如果纰漏， ... 博文

 教你修复u盘损坏了也可以修复 u盘修复工具修复工具

《奇巧淫技》系列-python！！每天早上八点自动发送天气预报邮件到QQ邮箱

阅读数 2万+

将代码部署服务器，每日早上定时获取到天气数据，并发送到邮箱。也可以说是一个小型人工智障。 ... 博文

<p>Python实例大全（基于Python3.7.4）</p> <p>博客说明： 这是自己写的有关python语言的一篇综合博客。 只作为知识广度和编程技巧学习，不过于...</p>	<p>阅读量 1万+</p> <p>博文</p>
<p>腾讯算法面试题：64匹马8个跑道需要多少轮才能选出最快的四匹？</p> <p>昨天，有网友私信我，说去阿里面试，彻底的被打击到了。问了为什么网上大量使用ThreadLocal的源...</p>	<p>阅读量 6万+</p> <p>博文</p>
<p>面试官：你连RESTful都不知道我怎么敢要你？</p> <p>干货，2019 RESTful最贱实践</p>	<p>阅读量 10万+</p> <p>博文</p>
<p>刷了几千道算法题，这些我私藏的刷题网站都在这里了！</p> <p>遥想当年，机缘巧合入了 ACM 的坑，周边巨擘林立，从此过上了"天天被虐似死狗"的生活... 然而我是...</p>	<p>阅读量 8万+</p> <p>博文</p>
<p>致 Python 初学者</p> <p>欢迎来到"Python进阶"专栏！来到这里的每一位同学，应该大致上学习了很多 Python 的基础知识，正...</p>	<p>阅读量 17万+</p> <p>博文</p>
<p>【C++100问】深入理解理解顶层const和底层const</p> <p>专栏C++学习笔记 声明 1) 该文章整理自网上的大牛和相关专家无私奉献的资料，具体引用的资料请看...</p>	<p>阅读量 4078</p> <p>博文</p>
<p>YouTube排名第一的励志英文演讲《Dream(梦想)》</p> <p>I don't know what that dream is that you have, I don't care how disappointing it might have been as y...</p>	<p>阅读量 4万+</p> <p>博文</p>
<p>吐血推荐珍藏的Visual Studio Code插件</p> <p>作为一名Java工程师，由于工作需要，最近一个月一直在写NodeJS，这种经历可以说是一部辛酸史了...</p>	<p>阅读量 1万+</p> <p>博文</p>
<p>“狗屁不通文章生成器”登顶GitHub热榜，分分钟写出万字形式主义大作</p> <p>一、垃圾文字生成器介绍 最近在浏览GitHub的时候，发现了这样一个骨骼清奇的雷人项目，而且热度...</p>	<p>阅读量 14万+</p> <p>博文</p>
<p>程序员：我终于知道post和get的区别</p> <p>是一个老生常谈的话题，然而随着不断的学习，对于以前的认识有很多误区，所以还是需要不断地总结...</p>	<p>阅读量 21万+</p> <p>博文</p>
<p>"狗屁不通文章生成器"登顶GitHub热榜，分分钟写出万字形式主义大作</p> <p>GitHub 被誉为全球最大的同性交友网站，.....，陪伴我们已经走过 10+ 年时间，它托管了大量的软件...</p>	<p>阅读量 9783</p> <p>博文</p>
<p>《程序人生》系列-这个程序员只用了20行代码就拿了冠军</p> <p>你知道的越多，你不知道的越多 点赞再看，养成习惯GitHub上已经开源https://github.com/JavaFamily...</p>	<p>阅读量 6万+</p> <p>博文</p>
<p>加快推动区块链技术和产业创新发展，2019可信区块链峰会在京召开</p> <p>11月8日，由中国信息通信研究院、中国通信标准化协会、中国互联网协会、可信区块链推进计划联合...</p>	<p>阅读量 6万+</p> <p>博文</p>
<p>Python 植物大战僵尸代码实现(2):植物卡片选择和种植</p> <p>这篇文章要介绍的是： - 上方植物卡片栏的实现。 - 点击植物卡片，鼠标切换为植物图片。 - 鼠标移动...</p>	<p>阅读量 1万+</p> <p>博文</p>
<p>Python3.7黑帽编程——病毒篇（基础篇）</p> <p>引子 Hacker（黑客），往往被人们理解为只会用非法手段来破坏网络安全的计算机高手。但是，黑客...</p>	<p>阅读量 1万+</p> <p>博文</p>
<p>程序员把地府后台管理系统做出来了，还有3.0版本！12月7号最新消息：已在开发中有github地址</p> <p>第一幕：缘起 听说阎王爷要做个生死簿后台管理系统，我们派去了一个程序员..... 996程序员做的梦...</p>	<p>阅读量 17万+</p> <p>博文</p>
<p>网易云6亿用户音乐推荐算法</p> <p>网易云音乐是音乐爱好者的集聚地，云音乐推荐系统致力于通过 AI 算法的落地，实现用户千人千面的...</p>	<p>阅读量 5万+</p> <p>博文</p>

大学生生活这样过，校招 offer 飞来找

本篇我们来聊聊大学生生活如何度过，才能在校招中拿到 offer。

阅读数 1万+

[博文](#)

小白都能看得懂的java虚拟机内存模型

目录 一、虚拟机 二、虚拟机组成 1.栈 栈帧 2.程序计数器 3.方法区 对象组成 4.本地方法栈 5.堆 GC G...

阅读数 3万+

[博文](#)

shell脚本基础

shell简介：shell是一种脚本语言，可以使用逻辑判断、循环等语法，可以自定义函数，是系统命令的集...

阅读数 1万+

[博文](#)

8年经验面试官详解 Java 面试秘诀

作者 |胡书敏 责编 | 刘静 出品 | CSDN（ID：CSDNnews） 本人目前在一家知名外企担任架构师，而且...

阅读数 9万+

[博文](#)

面试官如何考察你的思维方式？

1.两种思维方式在求职面试中，经常会考察这种问题：北京有多少量特斯拉汽车？某胡同口的煎饼摊一...

阅读数 5万+

[博文](#)

腾讯“疯狂”开源！

作者 | 马超 责编 | 胡巍巍 出品 | CSDN（ID：CSDNnews） 近日，腾讯自研的万亿级分布式消息中间...

阅读数 3万+

[博文](#)

so easy！10行代码写个“狗屁不通”文章生成器

前几天，GitHub 有个开源项目特别火，只要输入标题就可以生成一篇长长的文章。背后实现代码一定...

阅读数 9万+

[博文](#)

知乎高赞：中国有什么拿得出手的开源软件产品？(整理自本人原创回答)

知乎高赞：中国有什么拿得出手的开源软件产品？ 在知乎上，有个问题问“中国有什么拿得出手的开源...

阅读数 5万+

[博文](#)

MySQL数据库总结

一、数据库简介 数据库(Database，DB)是按照数据结构来组织，存储和管理数据的仓库。典型特征：...

阅读数 7万+

[博文](#)

20行Python代码爬取王者荣耀全英雄皮肤

引言 王者荣耀大家都玩过吧，没玩过的也应该听说过，作为时下最火的手机MOBA游戏，咳咳，好像...

阅读数 12万+

[博文](#)

网络（8）-HTTP、Socket、TCP、UDP的区别和联系

TCP/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包...

阅读数 1万+

[博文](#)

张小龙-年薪近3亿的微信之父，他是如何做到的？

张小龙生于湖南邵东魏家桥镇， 家庭主要特点：穷。不仅自己穷，亲戚也都很穷，可以说穷以类聚。...

阅读数 10万+

[博文](#)

阿里靠什么武功秘籍渡过“双十一”的天量冲击

双十一大概会产生多大的数据量呢，可能大家没概念，举个例子央视拍了这么多年电视新闻节目，几十...


阅读数 4万+

[博文](#)

python json java mysql pycharm android linux json格式 c# 单例模式里面的属性 c# lic文件 c# 盘古分词 c# 文本框只能输入- c# 十进制数字转换字母 c#某个变量更改刷新 c#往json里添加数据 c# 创建原子变量 c#怎么调用api接口 c#调用mstsc

没有更多推荐了, [返回首页](#)

©2019 CSDN 皮肤主题: 大白 设计师: CSDN官方博客



bear_wr

TA的个人主页 >

原创

22

粉丝

45

获赞

80

评论

30

访问

26万+

等级:

博客 5

周排名: 10万+

积分: 1731

总排名: 3万+

关注

私信

最新文章

【转】深入浅出 TCP/IP 协议

Integer类源码解析之toString方法

MarkDown学习 (2)

Markdown学习 (1)

介绍Sublime3下两款Markdown插件

分类专栏



Exception android

5篇



环境搭配

2篇



android开发

23篇



git

1篇



java

13篇

展开

归档

2018年5月

1篇

2018年4月

1篇

2018年3月

3篇

2017年9月

1篇

2017年5月

1篇

2017年1月

1篇

2016年10月

3篇

2016年9月

3篇

展开

热门文章

mybatis——select、insert、update、delete

阅读数 88274

ResultMap详解

阅读数 48187

MyBatis——动态SQL讲解

阅读数 17594

DispatcherServlet详解

阅读数 12627

mac上使用genymotion

阅读数 11414

最新评论

ResultMap详解

lzc2644481789: 想知道<resultMap>中的一对多怎么实现的, 具体的, 一步一步的

mybatis——select、i...

Robin_hc: 很棒!

ResultMap详解

weixin_44686337: niubi

mybatis——select、i...

qq_38928580: <insert>标签可以返回插入了几行数据么大佬?

mybatis——select、i...

qq_38928580: <update>标签好像也可以用来写 delete删除啊大佬? 这是啥情况? 用<update>写删 ...

优化 Mac 的性能

清理 Mac 并优化系统,
Mac 宛如新机



程序人生



CSDN资讯

👤 QQ客服

✉ kefu@csdn.net

🗣 客服论坛

☎ 400-660-0108

🕒 工作时间 8:30-22:00

关于我们 招聘 广告服务 网站地图

京ICP备19004658号 经营性网站备案信息

🚓 公安备案号 11010502030143

©1999-2020 北京创新乐知网络技术有限公司

网络110报警服务

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护 版权申诉