

 码龄7年  暂无认证

120

原创

1万+

周排名

169万+

总排名

49万+

访问



等级

4666

积分

276

粉丝

241

获赞

121

评论

407

收藏







私信

关注

搜博主文章

Q

- 热门文章
- 深入理解读写锁—ReadWriteLock源码分析

58556
- 深入理解Semaphore

49501
- Android开发——Snackbar使用详解

32166
- 深入理解Callable

25712
- 使用OkHttp进行网络同步异步操作

25501

- 分类专栏
-  OkHttp框架解析

5篇
-  Android基础

20篇
-  Android官方文档译文
-  深入浅出Android

4篇
-  Java集合库源码解析

9篇
-  View总结

4篇

- 最新评论
- 使用camera2构建相机应用

轨迹_Mine: 瓦的为啥报这个错-Access denied finding property "persist.vendor.sf.fbo..."
- 深入理解阻塞队列（二）——ArrayBlocki...

qq_16992475: happens-before是吧
- Flutter状态管理(2)——单Stream和广播S...

自带BGM的青春: 冒昧问下显示菊花是什么意思
- Flutter数据传输

ctotalk: 谢谢分享，不错
- 深入理解Callable

专注写bug: 文章写的很nice，差不多针对执行流程包括为什么会造成阻塞都有说明，...

您愿意向朋友推荐“博客详情页”吗？

原创

xingfeng_coder

2017-07-26 20:56:45

1484

收藏 3

版权

分类专栏:

Java集合库源码解析

文章标签:

hashtable

源码

线程安全

dictionary

 Java集合库源码... 专栏收录该内容

3 订阅

9 篇文章

订阅专栏

Hashtable和HashMap一样，都是一个哈希表，不允许键和值为null，该类是一个线程安全的，每个方法都加了synchronized关键字。下面是该类的继承关系图：
hashMap允许key和value为空

Class Hashtable<K,V>

```
java.lang.Object
java.util.Dictionary<K,V>
java.util.Hashtable<K,V>
```

All Implemented Interfaces:

Serializable, Cloneable, Map<K,V>

http://blog.csdn.net/qq_19431333

从上图可以看到，Hashtable继承自Dictionary类，而HashMap继承自AbstractMap，所以这两个类的祖宗就是不一样的。这篇文章主要介绍Hashtable和HashMap的异同点。对于HashMap不了解的朋友可以参考下面两篇文章：

- JDK1.8 HashMap源码分析
- JDK1.7 HashMap源码分析

构造器

底层结构

JDK1.8中HashMap的底层结构是数组+链表+红黑树，JDK1.7中HashMap的底层结构是数组+链表；而Hashtable的底层结构是数组+链表，本文的源码均基于JDK.1.8进行分析。由于Hashtable和JDK1.7中的HashMap都采用了数组+链表的结构，那么本文将以JDK1.8中的Hashtable和JDK1.7中的HashMap进行比较相同和不同的地方。


初始容量和加载因子


Hashtable和HashMap一样，都有初始容量和加载因子两个影响性能的参数，并且加载因子默认也是0.75。


构造方法

Hashtable的构造方法如下：

```
1 public Hashtable(int initialCapacity, float loadFactor) {
2     if (initialCapacity < 0)
3         throw new IllegalArgumentException("Illegal Capacity: "+
4                                           initialCapacity);
5     if (loadFactor <= 0 || Float.isNaN(loadFactor))
6         throw new IllegalArgumentException("Illegal Load: "+loadFactor);
7
8     if (initialCapacity==0)
9         initialCapacity = 1;
10    this.loadFactor = loadFactor;
11    table = new Entry<?,?>[initialCapacity];
12    threshold = (int)Math.min(initialCapacity * loadFactor, MAX_ARRAY_SIZE + 1);
13 }
14
15
16 public Hashtable(int initialCapacity) {
17     this(initialCapacity, 0.75f);
18 }
```

 疯狂盲盒





举报

取新又早

闲话元注解@Retention

2019年年终总结

ArrayMap数据结构分析

2020年 2篇

2019年 39篇

2018年 15篇

2017年 47篇

2016年 18篇

目录

构造器

底层结构

初始容量和加载因子

构造方法

基本操作

put(K k,V v)

get(K k)操作

remove(Object o)操作

迭代器

总结

```
22         this(11, 0.75f);
23     }
24
25
26     public Hashtable(Map<? extends K, ? extends V> t) {
27         this(Math.max(2*t.size(), 11), 0.75f);
28         putAll(t);
29     }
```

可以看到，Hashtable和HashMap的构造方法相同的是，均是对初始容量和加载因子完成了设置；不同的地方有2点：

1. HashMap对底层数组采取的懒加载，即当执行第一次插入时才会创建数组；而Hashtable在初始化时就创建了数组；
2. HashMap中数组的默认初始容量是16，并且必须的是2的指数倍数；而Hashtable中默认的初始容量是11，并且不要求必须是2的指数倍数。

基本操作

Hashtable作为哈希表，基本操作有插入一个键值对、按照键查询值以及删除键值对。下面逐个分析。


put(K k,V v)


put的实现如下：


```
1     public synchronized V put(K key, V value) {
2         //值不允许为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         //得到键的hash
10        int hash = key.hashCode();
11        //得到对应hash在数组中的桶索引
12        int index = (hash & 0x7FFFFFFF) % tab.length;
13        @SuppressWarnings("unchecked")
14        //得到桶中链表头节点
15        Entry<K,V> entry = (Entry<K,V>)tab[index];
16        //从头开始遍历
17        for(; entry != null ; entry = entry.next) {
18            //一旦hash值相等并且键相等，替换旧值
19            if ((entry.hash == hash) && entry.key.equals(key)) {
20                V old = entry.value;
21                entry.value = value;
22                return old;
23            }
24        }
25        //如果没有找到相同键，那么添加新节点
26        addEntry(hash, key, value, index);
27        return null;
28    }
```

下面看一下addEntry方法，其实现如下：

```
1     private void addEntry(int hash, K key, V value, int index) {
2         modCount++;
3
4         Entry<?,?> tab[] = table;
5         //如果尺寸超过了阈值，进行rehash
6         if (count >= threshold) {
```

疯狂盲盒





举报

```
12         index = (hash & 0x7FFFFFFF) % tab.length;
13     }
14
15     // Creates the new entry.
16     @SuppressWarnings("unchecked")
17     Entry<K,V> e = (Entry<K,V>) tab[index];
18     tab[index] = new Entry<>(hash, key, value, e);
19     count++;
20 }
```

从上面的代码可以看到，当插入一个节点时，如果哈希表的尺寸已经达到了扩容的阈值，那么进行rehash()，之后再

再将节点插入到链表的头部，这一点和HashMap是一样的，即新节点总是位于桶的头结点。

下面看一下rehash()方法， rehash()方法首先将数组扩容，然后再将数据从旧哈希表中移到新哈希表中，其实现如下：


```
1  protected void rehash() {
2      int oldCapacity = table.length;
3      Entry<?,?>[] oldMap = table;
4
5      // 扩容, newCapacity=2*oldCapacity+1
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     //rehash
20     for (int i = oldCapacity ; i-- > 0 ;) {
21         for (Entry<K,V> old = (Entry<K,V>)oldMap[i] ; old != null ; ) {
22             Entry<K,V> e = old;
23             old = old.next;
24
25             int index = (e.hash & 0x7FFFFFFF) % newCapacity;
26             e.next = (Entry<K,V>)newMap[index];
27             newMap[index] = e;
28         }
29     }
30 }
```


- rehash()方法主要分为两步：
1. 扩容。扩容策略为newCapacity=2*oldCapacity+1
 2. rehash。将节点rehash之后再当做头节点接到新的桶中

在上面的put方法中可以看到很多点与JDK1.7中不同的地方：

1. Hashtable的put()是线程安全的，而HashMap的put()方法不是线程安全的
2. HashMap中键和值均允许为null；Hashtable中均不允许
3. 计算hash的方式不同。Hashtable中使用键的哈希码作为哈希值，而HashMap中的哈希值将根据键的哈希值经过计算得到，其计算方式如下：

```
1  final int hash(Object k) {
2      int h = hashSeed;//默认为0
3      if (0 != h && k instanceof String) {
4          return sun.misc.Hashing.stringHash32((String) k);
5      }
```


疯狂盲盒





举报

```
10      // constant multiples at each bit position have a bounded
11      // number of collisions (approximately 8 at default load factor).
12      h ^= (h >>> 20) ^ (h >>> 12);
13      return h ^ (h >>> 7) ^ (h >>> 4);
14  }
```

并且HashMap中当hashSeed变化时，同一个键得到的hash值将会不一样。

4. 得到数组中桶的方式不一样。由于HashMap中桶的个数必须是2的指数倍数，因此得到桶索引处的方法为：

```
1  static int indexFor(int h, int length) {
2      // assert Integer.bitCount(length) == 1 : "length must be a non-zero power of 2";
3      return h & (length-1);
4  }
```

该方法就相当于对长度求模；而Hashtable中当hash值小于0x7FFFFFFF时和HashMap中一样，当大于0x7FFFFFFF时则不同。

5. 扩容策略。Hashtable扩容时策略是newCapacity=oldCapacity*2+1；而HashMap是newCapacity=2*oldCapacity

HashMap和Hashtable中put方法的相同点有如下2点：

1. 新节点总是作为桶的头节点
2. rehash时桶中的链表顺序会颠倒

get(K k)操作

Hashtable的get()方法用于根据键得到值，其实现如下：

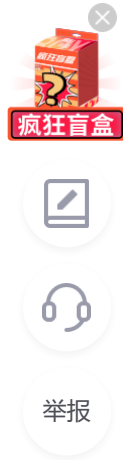
```
1  public synchronized V get(Object key) {
2      Entry<?,?> tab[] = table;
3      int hash = key.hashCode();
4      int index = (hash & 0x7FFFFFFF) % tab.length;
5      for (Entry<?,?> e = tab[index] ; e != null ; e = e.next) {
6          if ((e.hash == hash) && e.key.equals(key)) {
7              return (V)e.value;
8          }
9      }
10     return null;
11 }
```

可以看到该实现和HashMap是相同的，只不过是计算hash以及得到桶中索引的方式不同而已。、

remove(Object o)操作

Hashtable的remove()方法用于根据键删除键值对，其实现如下：

```
1  public synchronized V remove(Object key) {
2      Entry<?,?> tab[] = table;
3      //计算hash值
4      int hash = key.hashCode();
5      //得到桶的索引
6      int index = (hash & 0x7FFFFFFF) % tab.length;
7      @SuppressWarnings("unchecked")
8      Entry<K,V> e = (Entry<K,V>)tab[index];
9      //遍历
10     for(Entry<K,V> prev = null ; e != null ; prev = e, e = e.next) {
11         //如果匹配,修改节点
12         if ((e.hash == hash) && e.key.equals(key)) {
13             modCount++;
14             if (prev != null) {
15                 prev.next = e.next;
16             }
17         }
```




```
21         e.value = null;
22         return oldValue;
23     }
24 }
25 return null;
26 }
```

可以看到删除节点的操作是先计算hash，得到桶的索引，然后再遍历桶中的链表，这和HashMap中的实现一样。

迭代器

由于Hashtable没有实现Iterable接口，所以不能foreach循环遍历其键值，这是因为Hashtable从JDK1.0起就存在了，不过可以使用keys()方法得到键的集合，使用values()得到值的集合。keys()方法的实现如下：

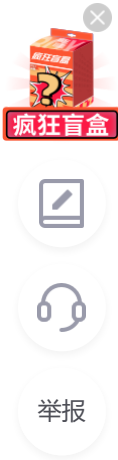
```
1 public synchronized Enumeration<K> keys() {
2     return this.<K>getEnumeration(KEYS);
3 }
```

其中Enumeration是一种类似于Iterator的接口，可以使用该类进行遍历。下面看一下getEnumeration(int type)方法，其实现如下：

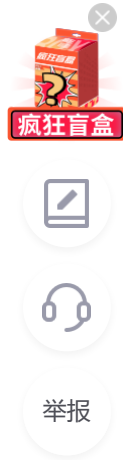
```
1 private <T> Enumeration<T> getEnumeration(int type) {
2     if (count == 0) {
3         return Collections.emptyEnumeration();
4     } else {
5         return new Enumerator<>(type, false);
6     }
7 }
```

可以看到，在哈希表不为空时，返回Enumerator对象，该类的定义如下：

```
1 private class Enumerator<T> implements Enumeration<T>, Iterator<T> {
2     Entry<?,?>[] table = Hashtable.this.table;
3     int index = table.length;
4     Entry<?,?> entry;
5     Entry<?,?> lastReturned;
6     int type;
7
8     /**
9      * Indicates whether this Enumerator is serving as an Iterator
10     * or an Enumeration. (true -> Iterator).
11     */
12     boolean iterator;
13
14     /**
15      * The modCount value that the iterator believes that the backing
16      * Hashtable should have. If this expectation is violated, the iterator
17      * has detected concurrent modification.
18     */
19     protected int expectedModCount = modCount;
20
21     Enumerator(int type, boolean iterator) {
22         this.type = type;
23         this.iterator = iterator;
24     }
25
26     public boolean hasMoreElements() {
27         Entry<?,?> e = entry;
28         int i = index;
29         Entry<?,?>[] t = table;
```



```
35         index = i;
36         return e != null;
37     }
38
39     @SuppressWarnings("unchecked")
40     public T nextElement() {
41         Entry<?,?> et = entry;
42         int i = index;
43         Entry<?,?>[] t = table;
44         /* Use locals for faster loop iteration */
45         while (et == null && i > 0) {
46             et = t[--i];
47         }
48         entry = et;
49         index = i;
50         if (et != null) {
51             Entry<?,?> e = lastReturned = entry;
52             entry = e.next;
53             return type == KEYS ? (T)e.key : (type == VALUES ? (T)e.value : (T)e);
54         }
55         throw new NoSuchElementException("Hashtable Enumerator");
56     }
57
58     // Iterator methods
59     public boolean hasNext() {
60         return hasMoreElements();
61     }
62
63     public T next() {
64         if (modCount != expectedModCount)
65             throw new ConcurrentModificationException();
66         return nextElement();
67     }
68
69     public void remove() {
70         if (!iterator)
71             throw new UnsupportedOperationException();
72         if (lastReturned == null)
73             throw new IllegalStateException("Hashtable Enumerator");
74         if (modCount != expectedModCount)
75             throw new ConcurrentModificationException();
76
77         synchronized(Hashtable.this) {
78             Entry<?,?>[] tab = Hashtable.this.table;
79             int index = (lastReturned.hash & 0x7FFFFFFF) % tab.length;
80
81             @SuppressWarnings("unchecked")
82             Entry<K,V> e = (Entry<K,V>)tab[index];
83             for(Entry<K,V> prev = null; e != null; prev = e, e = e.next) {
84                 if (e == lastReturned) {
85                     modCount++;
86                     expectedModCount++;
87                     if (prev == null)
88                         tab[index] = e.next;
89                     else
90                         prev.next = e.next;
91                     count--;
92                     lastReturned = null;
93                     return;
94                 }
95             }
96             throw new ConcurrentModificationException();
97         }
98     }
99 }
```



Enumeration接口的方法：

```
1 public interface Enumeration<E> {
2
3     boolean hasMoreElements();
4
5
6     E nextElement();
7 }
```

而Iterator接口的定义如下：

```
1 public interface Iterator<E> {
2
3     boolean hasNext();
4
5
6     E next();
7
8     default void remove() {
9         throw new UnsupportedOperationException("remove");
10    }
11 }
```

可以看到该两个接口基本是一致的。在Enumeration的实现中可以发现，除了remove()方法，Iterator接口的另外两个方法都是使用的Enumeration接口的实现，而remove()方法只有在iterator参数为true时才能使用，否则抛出异常。在keys()的调用过程中可以发现传入的iterator这个参数为false，那么什么时候这个参数会为true呢？

在使用values()方法得到值的集合时，iterator参数会为true，答案如下：

```
1 public Collection<V> values() {
2     if (values==null)
3         values = Collections.synchronizedCollection(new ValueCollection(),
4                                                         this);
5     return values;
6 }
```

由于values()的返回值是一个Collection，必须支持foreach遍历，并且由于Hashtable是线程安全的，所以values使用了Collections.synchronziedCollection()方法对ValueCollection就行了同步封装。ValueCollection类的定义如下：

```
1 private class ValueCollection extends AbstractCollection<V> {
2     public Iterator<V> iterator() {
3         return getIterator(VALUEES);
4     }
5     public int size() {
6         return count;
7     }
8     public boolean contains(Object o) {
9         return containsValue(o);
10    }
11    public void clear() {
12        Hashtable.this.clear();
13    }
14 }
```

主要关注iterator()方法，内部调用了getlterator()方法，该方法如下：

```
1 private <T> Iterator<T> getIterator(int type) {
2     if (count == 0) {
```

xingfeng_coder

关注


1


1


3



专栏目录

疯狂盲盒





举报

可以看到这时Enumerator的第二个参数为true。

总结

本文的Hashtable的代码是基于JDK1.8的，而与之比较的是1.7中的HashMap,因为它们的底层结构都是数组+链表。虽然大的结构上两个类相同，但是还是有主要的几点不同：

1. Hashtable是线程安全的；而HashMap不是线程安全的
2. 构造器的区别。Hashtable默认初始容量为11，HashMap为16
3. put方法的区别，主要包括hash的计算，桶中索引的计算，rehash

hashTable源码解析

Y1230601的博客

579


概述 JDK 1.8中的**HashTable**是底层实现由"数组+链表"实现，相对于hashMap来说简单很多，而且他们两个最大的区别是**ha...**

java HashTable源码解析

qq_41786692的博客


320

一、概述1.**HashTable**也是一个散列表，它存储的内容是键值对（key-value）映射，与HashMap不同的是**HashTable**的key...



优质评论可以帮助作者获得更高权重

评论

twowaterli: 发现一个很靠谱的Java书单 [Java工程师的终极书单](http://www.tiantianbianma.com/java-book-list-al...l.html/) 4 年前 回复 ...

HashTable源码解析_Chents

7-15

HashTable源码解析 1.特点 **HashTable**可以键值对形式的数据,key和value 都不能为null。同时通过synchronized给put和get...

C# Hashtable源码剖析_exiaojiu的博客

9-15

1. key:键,键是不能重复的; 2. val:值,可以是任何的类型(想要类型安全可以选择**Dictionary**,是**Hashtable**的泛型实现); 3. hash_...

java hashtable 初始化_java hashtable 初始化为啥是11

weixin_34434843的博客

375

et = entry;int i = index;Entry[] t = table; /* Use locals for faster loop iteration */while (et == null && i > 0) {et = t[--i];entry = et;...

java hashtable_hashmap_java – 为什么Hashtable的initialCapacity为11,而Hash...weixin_35757732的博客

217

下面的文章详细解决了这个问题：HashMap requires a better hashCode() – JDK 1.4 Part II。根据该文章，移动到二的幂的...

Hashtable源码_紫陌。。。的博客

7-19

Hashtable基于哈希表实现的,每个元素都是key-value对,通过单链表解决哈希冲突。 **Hashtable**是**线程安全**的,实现Serializabl...

HashTable实现原理以及源码解析

Eric的博客

1万+

HashTable实现原理以及**源码**解析1、**HashTable** 和HashMap一样，**Hashtable** 也是一个散列表，它存储的内容是键值对(ke...

Hashtable源码解析（JDK1.8）

武培轩

163

package java.util; import java.io.*; import java.util.concurrent.ThreadLocalRandom; import java.util.function.BiConsumer; i...

Hashtable源码解析

qq_37254296的博客

91

Hashtable算是平时用的比较少的一个集合了，先从继承、实现关系 及 构造函数来简单的了解一下 public class **Hashtable**...

jdk1.8的Hashtable源码解析

qq_29864971的博客

1633

Hashtable声明public class **Hashtable**<K,V> extends **Dictionary**<K,V> implements Map<K,V>, Cloneable, jav...

jdk1.8中hashtable源码分析

YYC的专栏

1978

注：基于JDK 1.8.0_131源代码为例进行分析 **hashtable**的结构图 **hashtable**采用桶位+链表结构实现。 **hashtable**的实现 采...

java_集合体系之Hashtable详解、源码及示例——10

Oscar Chen

3039

摘要： 本文通过**Hashtable**的结构图来说明**Hashtable**的结构、以及所具有的功能。根据**源码**给出**Hashtable**所具有的特性、...

hash算法及Java的HashTable源码分析

tyhj_sf的博客空间

1120

本文详细讲解Hash算法原理及Java中**Hashtable**类的**源码**，因为**hashtable**类实的现正是使用了hash算法。

HashTable源码解析 最新发布

Jak的博客


30

一、**HashTable**的基本介绍 1.存放的元素是键值对： 即K-V 2.**hashtable**的键和值都不能为null，否则会抛出NullPointerExce...


weixin_35628777的博客

14

都是key-value对，其内部也是通过单链表...

疯狂盲盒





举报