

深入Java集合学习系列：HashSet的实现原理

博客分类：Core Java

原文地址：<http://zhangshixi.iteye.com/blog/673143>

Java

C

C++



C#

1. HashSet概述：

HashSet实现Set接口，由哈希表（实际上是一个HashMap实例）支持。它不保证set的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用null元素。

2. HashSet的实现：

对于HashSet而言，它是基于HashMap实现的，HashSet底层使用HashMap来保存所有元素，因此HashSet的实现比较简单，相关HashSet的操作，基本上都是直接调用底层HashMap的相关方法来完成，HashSet的源代码如下：

Java代码  

```
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.     // 底层使用HashMap来保存HashSet中所有元素。
8.     private transient HashMap<E, Object> map;
9.
10.    // 定义一个虚拟的Object对象作为HashMap的value，将此对象定义为static final。
11.    private static final Object PRESENT = new Object();
12.
13.    /**
14.     * 默认的空参构造器，构造一个空的HashSet。
15.     *
16.     * 实际底层会初始化一个空的HashMap，并使用默认初始容量为16和加载因子0.75。
17.     */
18.    public HashSet() {
19.        map = new HashMap<E, Object>();
20.    }
21.
22.    /**
```

```

23.      * 构造一个包含指定collection中的元素的新set。
24.      *
25.      * 实际底层使用默认的加载因子0.75和足以包含指定
26.      * collection中所有元素的初始容量来创建一个HashMap。
27.      * @param c 其中的元素将存放在此set中的collection。
28.      */
29.      public HashSet(Collection<? extends E> c) {
30.          map = new HashMap<E, Object>(Math.max((int) (c.size()/.75f) + 1, 16));
31.          addAll(c);
32.      }
33.
34.      /**
35.       * 以指定的initialCapacity和loadFactor构造一个空的HashSet。
36.       *
37.       * 实际底层以相应的参数构造一个空的HashMap。
38.       * @param initialCapacity 初始容量。
39.       * @param loadFactor 加载因子。
40.       */
41.      public HashSet(int initialCapacity, float loadFactor) {
42.          map = new HashMap<E, Object>(initialCapacity, loadFactor);
43.      }
44.
45.      /**
46.       * 以指定的initialCapacity构造一个空的HashSet。
47.       *
48.       * 实际底层以相应的参数及加载因子loadFactor为0.75构造一个空的HashMap。
49.       * @param initialCapacity 初始容量。
50.       */
51.      public HashSet(int initialCapacity) {
52.          map = new HashMap<E, Object>(initialCapacity);
53.      }
54.
55.      /**
56.       * 以指定的initialCapacity和loadFactor构造一个新的空链接哈希集合。
57.       * 此构造函数为包访问权限，不对外公开，实际只是是对LinkedHashSet的支持。
58.       *
59.       * 实际底层会以指定的参数构造一个空LinkedHashMap实例来实现。
60.       * @param initialCapacity 初始容量。
61.       * @param loadFactor 加载因子。
62.       * @param dummy 标记。
63.       */
64.      HashSet(int initialCapacity, float loadFactor, boolean dummy) {

```

```

65.     map = new LinkedHashMap<E, Object>(initialCapacity, loadFactor);
66. }
67.
68. /**
69.  * 返回对此set中元素进行迭代的迭代器。返回元素的顺序并不是特定的。
70.  *
71.  * 底层实际调用底层HashMap的keySet来返回所有的key。
72.  * 可见HashSet中的元素，只是存放在了底层HashMap的key上，
73.  * value使用一个static final的Object对象标识。
74.  * @return 对此set中元素进行迭代的Iterator。
75.  */
76. public Iterator<E> iterator() {
77.     return map.keySet().iterator();
78. }
79.
80. /**
81.  * 返回此set中的元素的数量（set的容量）。
82.  *
83.  * 底层实际调用HashMap的size()方法返回Entry的数量，就得到该Set中元素的个数。
84.  * @return 此set中的元素的数量（set的容量）。
85.  */
86. public int size() {
87.     return map.size();
88. }
89.
90. /**
91.  * 如果此set不包含任何元素，则返回true。
92.  *
93.  * 底层实际调用HashMap的isEmpty()判断该HashSet是否为空。
94.  * @return 如果此set不包含任何元素，则返回true。
95.  */
96. public boolean isEmpty() {
97.     return map.isEmpty();
98. }
99.
100. /**
101.  * 如果此set包含指定元素，则返回true。
102.  * 更确切地讲，当且仅当此set包含一个满足(o==null ? e==null : o.equals(e))
103.  * 的e元素时，返回true。
104.  *

```

```

105.     * 底层实际调用HashMap的containsKey判断是否包含指定key。
106.     * @param o 在此set中的存在已得到测试的元素。
107.     * @return 如果此set包含指定元素，则返回true。
108.     */
109.     public boolean contains(Object o) {
110.         return map.containsKey(o);
111.     }
112.
113.     /**
114.     * 如果此set中尚未包含指定元素，则添加指定元素。
115.     * 更确切地讲，如果此 set 没有包含满足(e==null ? e2==null : e.equals(e2))
116.     * 的元素e2，则向此set 添加指定的元素e。
117.     * 如果此set已包含该元素，则该调用不更改set并返回false。
118.     *
119.     * 底层实际将该元素作为key放入HashMap。
120.     * 由于HashMap的put()方法添加key-value对时，当新放入HashMap的Entry中key
121.     * 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），
122.     * 新添加的Entry的value会将覆盖原来Entry的value，但key不会有任何改变，
123.     * 因此如果向HashSet中添加一个已经存在的元素时，新添加的集合元素将不会被放入HashMap中，
124.     * 原来的元素也不会有任何改变，这也就满足了Set中元素不重复的特性。
125.     * @param e 将添加到此set中的元素。
126.     * @return 如果此set尚未包含指定元素，则返回true。
127.     */
128.     public boolean add(E e) {
129.         return map.put(e, PRESENT)==null;
130.     }
131.
132.     /**
133.     * 如果指定元素存在于此set中，则将其移除。
134.     * 更确切地讲，如果此set包含一个满足(o==null ? e==null : o.equals(e))的元素e，
135.     * 则将其移除。如果此set已包含该元素，则返回true
136.     * （或者：如果此set因调用而发生更改，则返回true）。（一旦调用返回，则此set不再包含该元素）。
137.     *
138.     * 底层实际调用HashMap的remove方法删除指定Entry。
139.     * @param o 如果存在于此set中则需要将其移除的对象。
140.     * @return 如果set包含指定元素，则返回true。
141.     */
142.     public boolean remove(Object o) {
143.         return map.remove(o)==PRESENT;
144.     }
145.
146.     /**
147.     * 从此set中移除所有元素。此调用返回后，该set将为空。
148.     *

```

```

149.     * 底层实际调用HashMap的clear方法清空Entry中所有元素。
150.     */
151.     public void clear() {
152.         map.clear();
153.     }
154.
155.     /**
156.     * 返回此HashSet实例的浅表副本：并没有复制这些元素本身。
157.     *
158.     * 底层实际调用HashMap的clone()方法，获取HashMap的浅表副本，并设置到HashSet中。
159.     */
160.     public Object clone() {
161.         try {
162.             HashSet<E> newSet = (HashSet<E>) super.clone();
163.             newSet.map = (HashMap<E, Object>) map.clone();
164.             return newSet;
165.         } catch (CloneNotSupportedException e) {
166.             throw new InternalError();
167.         }
168.     }
169. }

```

3. 相关说明：

- 1) 相关HashMap的实现原理，请参考我的上一遍总结：[深入Java集合学习系列：HashMap的实现原理](#)。
- 2) 对于HashSet中保存的对象，请注意正确重写其equals和hashCode方法，以保证放入的对象的唯一性。

contains 方法中：为什么List中的对象实现 equals 即可，而Set中的对象需要实现 equals & hashCode？

List 是用一个Object [] data 来存储数据，通过循环找出目标对象，然后调用equals即可。

Set 是由HashMap 实现，HashMap 中用Entry[] table来存储 hash(key) 数据，如果key的hash有重复，通过 Entry中的next来查询对象，所以需要实现 hash . equals

