

HashSet实现原理分析（Java源码剖析）



六尺帐篷 (/u/f8e9b1c246f1)

2017.08.06 14:25 字数 1005 阅读 104 评论 0 喜欢 6

(/u/f8e9b1c246f1)

[编辑文章 \(/writer#/notebooks/15099409/notes/15463390\)](#)

本文将深入讨论HashSet实现原理的源码细节。在分析源码之前，首先我们需要对HashSet有一个基本的理解。

- HashSet只存储不同的值，set中是不会出现重复值的。
- HashSet和HashMap一样也需要实现hash算法来计算对象的hash值，但不同的是，HashMap中添加一个键值对的时候，(Key, Value)，hash函数计算的是Key的hash值。而HashSet则是计算value的hash值。当我们调用HashSet的add (E e) 的方法的时候，我们会计算元素e的hash值，如果这个值之前没出现过，就说明这个元素在set中不存在，如果出现过，就说明。set中已经存在了，就添加失败。

知道了上述的基本概念之后，我们就可以打开JDK源码，来一探究竟了。

关于hashSet的实现原理，最重要的一个点就是**HashSet内部是使用HashMap来存储对象的**。所以请读者务必先对hashMap的实现原理有一个初步的认识。参考笔者的文章HashMap实现原理分析（Java源码剖析）(<http://www.jianshu.com/p/6acb078ee895>)

我们可以看到HashSet有多个构造函数，但每个构造函数都是初始化了一个HashMap的对象



```
/**
 * Constructs a new, empty set; the backing <tt>HashMap</tt> instance has
 * default initial capacity (16) and load factor (0.75).
 */
public HashSet() {
    map = new HashMap<>();
}

/**
 * Constructs a new set containing the elements in the specified
 * collection. The <tt>HashMap</tt> is created with default load factor
 * (0.75) and an initial capacity sufficient to contain the elements in
 * the specified collection.
 *
 * @param c the collection whose elements are to be placed into this set
 * @throws NullPointerException if the specified collection is null
 */
public HashSet(Collection<? extends E> c) {
    map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
    addAll(c);
}

/**
 * Constructs a new, empty set; the backing <tt>HashMap</tt> instance has
 * the specified initial capacity and the specified load factor.
 *
 * @param    initialCapacity    the initial capacity of the hash map
 * @param    loadFactor         the load factor of the hash map
 * @throws   IllegalArgumentException if the initial capacity is less
 *          than zero, or if the load factor is nonpositive
 */
public HashSet(int initialCapacity, float loadFactor) {
    map = new HashMap<>(initialCapacity, loadFactor);
}

/**
 * Constructs a new, empty set; the backing <tt>HashMap</tt> instance has
 * the specified initial capacity and default load factor (0.75).
 *
 * @param    initialCapacity    the initial capacity of the hash table
 * @throws   IllegalArgumentException if the initial capacity is less
 *          than zero
 */
public HashSet(int initialCapacity) {
    map = new HashMap<>(initialCapacity);
}
```

我们可以观察到，默认的构造函数指定的初始化容量是16，负载因子是0.75。。也就是说创建了一个长度为16的数组，默认的负载因子为0.75，当达到容量时，map会自动扩容。

而这里的HashMap的是如下：

```
private transient HashMap<E,Object> map;
```

可以看到，HashSet中使用的HashMap，key为Set的元素类型，value为Object。

add(E e)

我们来看add方法的实现



```
/**
 * Adds the specified element to this set if it is not already present.
 * More formally, adds the specified element e to this set if
 * this set contains no element e2 such that
 * (e==null&&e2==null&&e.equals(e2)).
 * If this set already contains the element, the call leaves the set
 * unchanged and returns false.
 *
 * @param e element to be added to this set
 * @return true if this set did not already contain the specified
 * element
 */
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

PRESENT的定义

```
// Dummy value to associate with an Object in the backing Map
private static final Object PRESENT = new Object();
```

我们看到PRESENT是一个静态的类对象，Object类型。所有HashSet的实例都共享这个对象。

也就是说，我们在向set中添加一个e元素的时候，实际上就是在像map添加一个（e，Object）的键值对。我们添加的元素e变成了map中的key，而value则都是Object对象。又因为map中key值是唯一的，而value是可以重复的。所以我们添加的e作为key的话，就可以保证添加成功的话，e一定是唯一的。这就实现了set的唯一性。

remove(Object o)

我们接下来看remove的代码

```
/**
 * Removes the specified element from this set if it is present.
 * More formally, removes an element e such that
 * (o==null&&e==null&&o.equals(e)),
 * if this set contains such an element. Returns true if
 * this set contained the element (or equivalently, if this set
 * changed as a result of the call). (This set will not contain the
 * element once the call returns.)
 *
 * @param o object to be removed from this set, if present
 * @return true if the set contained the specified element
 */
public boolean remove(Object o) {
    return map.remove(o)==PRESENT;
}
```

我们知道HashMap中移除一个key的话，会返回这个key值对应的value，而我们这里的map，所有的key的value都是同一个对象PRESENT，所以我们这里只需要判断map.remove(o)的返回值是不是PRESENT，就可以确定是否成功移除了

iterator()

我们知道HashSet没有get方法，想要获取HashSet的元素需要调用iterator()

为什么会这样呢？

其实只要我们结合HashSet底层是由HashMap实现的就知道，我们添加的元素值都被map当成了key来存储，显然没有从map中获取单独一个key的方法，但是我们可以获取所有key，调用keySet方法即可。



```
/**
 * Returns an iterator over the elements in this set. The elements
 * are returned in no particular order.
 *
 * @return an Iterator over the elements in this set
 * @see ConcurrentModificationException
 */
public Iterator<E> iterator() {
    return map.keySet().iterator();
}
```

小结

HashSet由于内部实现是完全基于HashMap的，所以原理较为简单，代码也不多，源码加上注释也就是300多行。

关于hashSet的实现原理，我们需要掌握一下几点


- hashSet内部用HashMap来存储元素
- HashSet利用本身的值来计算hash值，因为值被当作hashmap的key，而hashmap是利用key来计算hash值的
- 因为hashset将value当作key来存储，所以根据map的key值唯一的原理，我们就可以实现set的无重复元素的功能



六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字，被 15240 人关注，获得了 1429 个喜欢

(/u/f8e9b1c246f1)

 喜欢

6







更多分享

(http://cwb.assets.jianshu.io/notes/images/1546339

被以下专题收入，发现更多相似内容

投稿管理

- + 收入我的专题
- 

Android... (/c/5139d555c94d?utm_source=desktop&utm_medium=notes-included-collection)
- 

Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)
- 

Android... (/c/58b4c20abf2f?utm_source=desktop&utm_medium=notes-included-collection)
- 

Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)
- 

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)
- 


Android开发 (/c/213623b75a3a?utm_source=desktop&utm_medium=notes-







included-collection)

 java (/c/4f285596f858?utm_source=desktop&utm_medium=notes-included-collection)

included-collection)

