HashSet源码阅读

概述

实际上，是基于HashMap的实现。

HashSet底层使用HashMap来保存所有元素。HashSet的操作基本上都是直接调用底层HashMap的相关方法来完成。

总结：

- 非线程安全
- 允许出现null值（HashMap中允许出现null的key，和null的value）
- 保证唯一性：先去获取对象的hash值，如果值一样，再调用equals比较，如果也一样，那就认为是一样的，那么不add元素

  所以，需要正确写好hashCode方法和equals方法

1. 类头

```
public class HashSet<E> extends AbstractSet<E>
    implements Set<E>, Cloneable, java.io.Serializable
```

继承了`AbstractSet`

实现了`Set`，Set继承了Collection，但是没有对其进行扩展，只是概念上限制了里面的元素不能出现重复

2. 实例变量 & 静态变量

```
private transient HashMap<E,Object> map;        // 存储hashSet的数据
```

——可以发现，hashSet本质上是用哈希表进行存储的

静态常量，主要用在存储中，由于底层是HashMap，需要有key-value，一对一对的存储，而hashSet只有一个值，所以我们**人为补value。**

```
private static final Object PRESENT = new Object();     // 一个空的object对象
```

3. 构造方法

无参构造方法

```
public HashSet() {
    map = new HashMap<>();          // 初始化一个哈希表
}
```

带参构造方法

传递初始化容量和负载因子——本质上就是传递给hashMap使用的。

```
public HashSet(int initialCapacity, float loadFactor) {
    map = new HashMap<>(initialCapacity, loadFactor);
}
```

只传递初始容量

```
public HashSet(int initialCapacity) {
    map = new HashMap<>(initialCapacity);
}
```

直接传递一个集合对象过去，默认负载因子还是0.75，然后根据这个默认负载因子去创建足够长度的初始容量

```
public HashSet(Collection<? extends E> c) {
    map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
    addAll(c);
}
```

4. 实例方法

大部分都是直接调用hashMap的方法，这边只是列出可以有哪些方法：

```
public int size();
public boolean isEmpty();
// 和hashMap有不同的地方
public boolean contains(Object o);      //--注意这边是contains，实现就是去
hashMap.containsKey(xxx)

// 添加元素--这边的value就是一个虚拟的值
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}

public boolean remove(Object o) {
    return map.remove(o)==PRESENT;
}

public void clear()
```

理解：

1. 这边的add，就是**先去判断哈希值**、哈希值一样接着判断是否是同一对象 or 不是同一对象 但是equals值一样——这样才能完全判断一个key是否出现重复
2. 删除，这边会根据返回值来判断是否删除成功

`clone()` 方法：

```
public Object clone() {
    try {
        HashSet<E> newSet = (HashSet<E>) super.clone();
        newSet.map = (HashMap<E, Object>) map.clone();      // 实际上是调用
hashMap.clone方法
        return newSet;
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}
```

参考:

1. https://blog.csdn.net/fighterandknight/article/details/66585997