一、基础概念

1. 基本介绍

TreeSet是有序的，与HashSet和HashMap的关系一样，TreeSet是基于TreeMap的。

2. 构造方法

(1) public TreeSet() {

this(new TreeMap<E,Object>());

} //默认构造方法假定元素实现了Comparable接口

public TreeSet(Comparator<? super E> comparator) {

this(new TreeMap<>(comparator));

} //使用传入的比较器，不要求元素实现Comparable。

说明： TreeSet实现了排重和排序，排重是指TreeMap中key值不能出现重复，排序是红黑树实现的。

3. 底层实现原理

3.1 基本属性

private static final Object PRESENT = new Object(); //固定的value值

private transient NavigableMap<E,Object> m; //底层的TreeMap

3.2 添加元素add

public boolean add(E e) {

//调用TreeMap中的put()方法，如果添加一个新节点则返回null

return m.put(e, PRESENT)==null;

}

3.3 删除元素remove

public boolean remove(Object o) {

//调用相应的方法，根据key去删除元素

return m.remove(o)==PRESENT;

}

3.4 子集视图

public SortedSet<E> subSet(E fromElement, E toElement) {

//获取一个起始元素到终止元素的子集视图

return subSet(fromElement, true, toElement, false);

}

3.5 首尾操作

(1) public E first() {

//最左边的那个元素

return m.firstKey();

}

(2) public E last() {

//最右边的那个元素

return m.lastKey();

}

(3) public E pollFirst() {

//删除最左边的那个元素

Map.Entry<E,?> e = m.pollFirstEntry();

return (e == null) ? null : e.getKey();

}

(4) public E pollLast() {

//删除最右边的那个元素

Map.Entry<E,?> e = m.pollLastEntry();

return (e == null) ? null : e.getKey();

}

二、总结

1. 没有重复元素

2. 添加、删除元素、判断元素是否存在，效率比较高，为O(log2(N))，N为元素个数。

3. 有序，TreeSet同样实现了SortedSet和NavigatableSet接口，可以方便的根据顺序进行查找和操作，如第一个、最后一个、某一取值范围、某一值的邻近元素等。

4. 为了有序，TreeSet要求元素实现Comparable接口或通过构造方法提供一个Comparator对象。