芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2019-12-16
JDK

精尽 JDK 源码解析 —— 集合（七）TreeSet

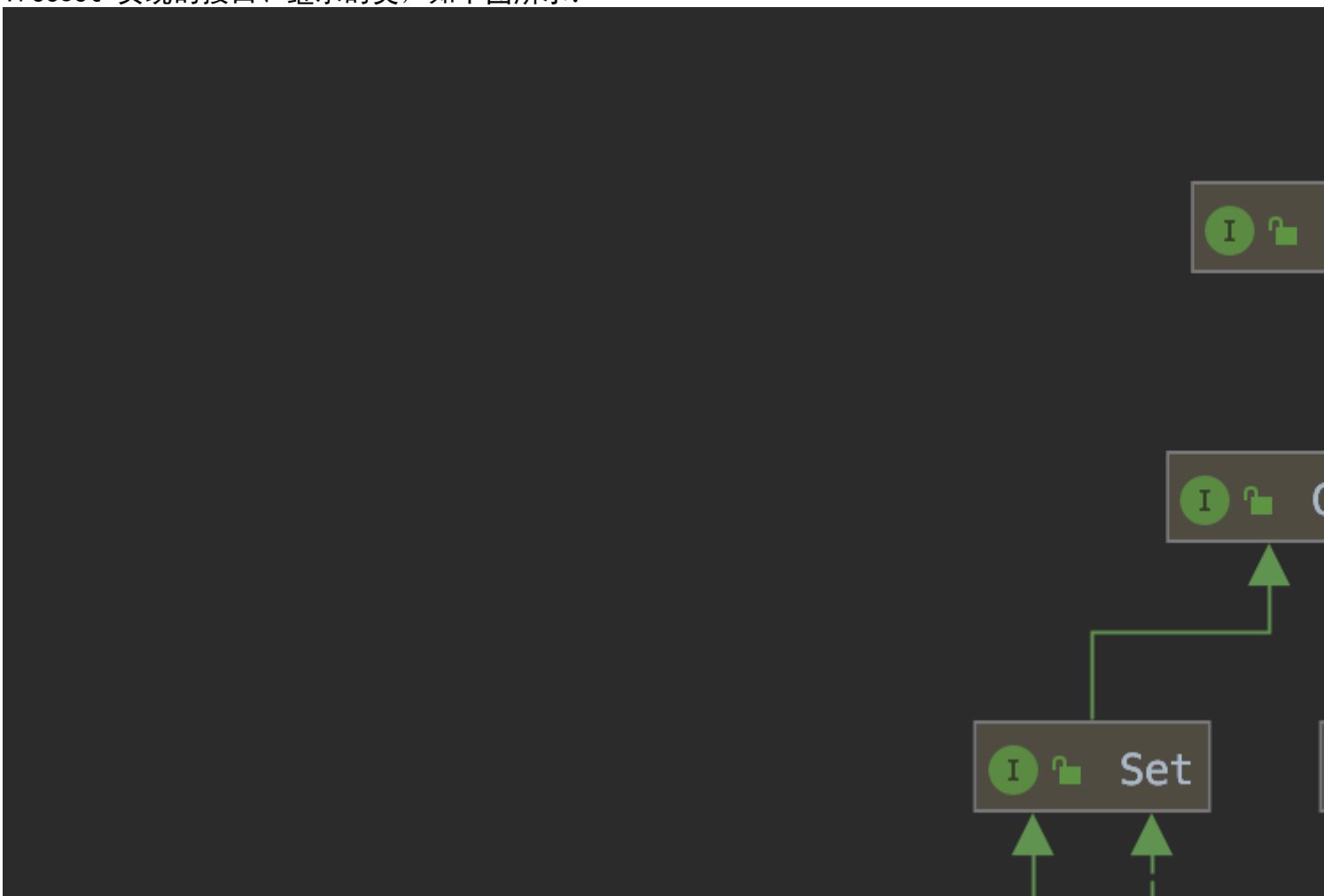
1. 概述

本文，和 《精尽 JDK 源码解析 —— 集合（五）哈希集合 HashSet》 基本是一致的。

TreeSet ，基于 TreeSet 的 Set 实现类。在业务中，如果我们有排重+ 排序的需求，一般会考虑使用 TreeSet 。不过，貌似很少会出现排重+ 排序的双重需求。所以呢，TreeSet 反正芳芳是木有使用过。

2. 类图

TreeSet 实现的接口、继承的类，如下图所示：



实现 java.util.NavigableSet 接口，并继承 java.util.AbstractSet 抽像类。
实现 java.io.Serializable 接口。
实现 java.lang.Cloneable 接口。

对于 NavigableSet 和 SortedMap 接口，已经添加注释，胖友可以直接点击查看。

3. 属性

TreeSet 只有一个属性，那就是 m 。代码如下：

```
// TreeSet.java

private transient NavigableMap<E,Object> m;
```

m 的 key ，存储 HashSet 的每个 key 。

map 的 value ，因为 TreeSet 没有 value 的需要，所以使用一个统一的 PRESENT 即可。代码如下：

```
// TreeSet.java

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

4. 构造方法

TreeSet 一共有 5 个构造方法，代码如下：

```
// TreeSet.java

TreeSet(NavigableMap<E,Object> m) {
    this.m = m;
}

public TreeSet() {
    this(new TreeMap<>());
}

public TreeSet(Comparator<? super E> comparator) {
    this(new TreeMap<>(comparator));
}

public TreeSet(Collection<? extends E> c) {
    this();
    // 批量添加
    addAll(c);
}

public TreeSet(SortedSet<E> s) {
    this(s.comparator());
    // 批量添加
```

```
        addAll(s);
    }
```

在构造方法中，会创建 TreeMap 对象，赋予到 m 属性。

5. 添加单个元素

#add(E e) 方法，添加单个元素。代码如下：

```
    // TreeSet.java

    public boolean add(E e) {
        return m.put(e, PRESENT)==null;
    }
```

m 的 value 值，就是我们看到的 PRESENT 。

#addAll(Collection<? extends E> c) 方法，批量添加。代码如下：

```
    // TreeSet.java

    public  boolean addAll(Collection<? extends E> c) {
        // Use linear-time version if applicable
        // 情况一
        if (m.size()==0 && c.size() > 0 &&
            c instanceof SortedSet &&
            m instanceof TreeMap) {
            SortedSet<? extends E> set = (SortedSet<? extends E>) c;
            TreeMap<E,Object> map = (TreeMap<E, Object>) m;
            if (Objects.equals(set.comparator(), map.comparator())) {
                map.addAllForTreeSet(set, PRESENT);
                return true;
            }
        }
        // 情况二
        return super.addAll(c);
    }
```

在实现上，和 TreeMap 的批量添加是一样的，对于情况一，会进行优化。

6. 移除单个元素

#remove(Object o) 方法，移除 o 对应的 value ，并返回是否成功。代码如下：

```
    // TreeSet.java

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

7. 查找单个元素

#contains(Object key) 方法，判断 key 是否存在。代码如下：

```
// TreeSet.java

public boolean contains(Object o) {
    return m.containsKey(o);
}
```

芳芳：后面的内容，快速看即可。

8. 查找接近的元素

在 NavigableSet 中，定义了四个查找接近的元素：

#lower(E e) 方法，小于 e 的 key
#floor(E e) 方法，小于等于 e 的 key
#higher(E e) 方法，大于 e 的 key
#ceiling(E e) 方法，大于等于 e 的 key

我们一起来看看哈。

```
// TreeSet.java

public E lower(E e) {
    return m.lowerKey(e);
}

public E floor(E e) {
    return m.floorKey(e);
}

public E ceiling(E e) {
    return m.ceilingKey(e);
}

public E higher(E e) {
    return m.higherKey(e);
}
```

9. 获得首尾的元素

#first() 方法，获得首个 key 。代码如下：

```
// TreeSet.java

public E first() {
    return m.firstKey();
```

```
}
```

#pollFirst() 方法，获得并移除首个 key 。代码如下：

```
// TreeSet.java

public E pollFirst() {
    Map.Entry<E,?> e = m.pollFirstEntry();
    return (e == null) ? null : e.getKey();
}
```

#last() 方法，获得尾部 key 。代码如下：

```
// TreeSet.java

public E last() {
    return m.lastKey();
}
```

#pollLast() 方法，获得并移除尾部 key 。代码如下：

```
// TreeSet.java

public E pollLast() {
    Map.Entry<E,?> e = m.pollLastEntry();
    return (e == null) ? null : e.getKey();
}
```

10. 清空

#clear() 方法，清空。代码如下：

```
// TreeSet.java

public void clear() {
    m.clear();
}
```

11. 克隆

#clone() 方法，克隆 TreeSet 。代码如下：

```
// TreeSet.java

public Object clone() {
    // 克隆创建 TreeSet 对象
```

```
        TreeSet<E> clone;
        try {
            clone = (TreeSet<E>) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new InternalError(e);
        }

        // 创建 TreeMap 对象，赋值给 clone 的 m 属性
        clone.m = new TreeMap<>(m);
        return clone;
    }
```

12. 序列化

#writeObject(ObjectOutputStream s) 方法，序列化 TreeSet 对象。代码如下：

```
    // TreeSet.java

    @java.io.Serial
    private void writeObject(java.io.ObjectOutputStream s)
        throws java.io.IOException {
        // Write out any hidden stuff
        // 写入非静态属性、非 transient 属性
        s.defaultWriteObject();

        // Write out Comparator
        // 写入比较器
        s.writeObject(m.comparator());

        // Write out size
        // 写入 key-value 键值对数量
        s.writeInt(m.size());

        // Write out all elements in the proper order.
        // 写入具体的 key-value 键值对
        for (E e : m.keySet())
            s.writeObject(e);
    }
```

13. 反序列化

#readObject(ObjectInputStream s) 方法，反序列化成 TreeSet 对象。代码如下：

```
    // TreeSet.java

    @java.io.Serial
    private void readObject(java.io.ObjectInputStream s)
        throws java.io.IOException, ClassNotFoundException {
        // Read in any hidden stuff
        // 读取非静态属性、非 transient 属性
        s.defaultReadObject();
```

```
    // Read in Comparator
    // 读取比较器
    @SuppressWarnings("unchecked")
    Comparator<? super E> c = (Comparator<? super E>) s.readObject();

    // Create backing TreeMap
    // 创建 TreeMap 对象
    TreeMap<E,Object> tm = new TreeMap<>(c);
    m = tm;

    // Read in size
    // 读取 key-value 键值对数量
    int size = s.readInt();

    // 读取具体的 key-value 键值对
    tm.readTreeSet(size, s, PRESENT);
}

// TreeMap.java

void readTreeSet(int size, java.io.ObjectInputStream s, V defaultVal)
    throws java.io.IOException, ClassNotFoundException {
    buildFromSorted(size, null, s, defaultVal);
}
```

14. 获得迭代器

```
// TreeSet.java

public Iterator<E> iterator() { // 正序 Iterator 迭代器
    return m.navigableKeySet().iterator();
}

public Iterator<E> descendingIterator() { // 倒序 Iterator 迭代器
    return m.descendingKeySet().iterator();
}
```

15. 转换成 Set/Collection

```
// TreeSet.java

public NavigableSet<E> descendingSet() {
    return new TreeSet<>(m.descendingMap());
}
```

16. 查找范围的元素

```
// TreeSet.java
```

```
// subSet 组
public NavigableSet<E> subSet(E fromElement, boolean fromInclusive,
                              E toElement,   boolean toInclusive) {
    return new TreeSet<>(m.subMap(fromElement, fromInclusive,
                                  toElement,   toInclusive));
}
public SortedSet<E> subSet(E fromElement, E toElement) {
    return subSet(fromElement, true, toElement, false);
}

// headSet 组
public NavigableSet<E> headSet(E toElement, boolean inclusive) {
    return new TreeSet<>(m.headMap(toElement, inclusive));
}
public SortedSet<E> headSet(E toElement) {
    return headSet(toElement, false);
}

// tailSet 组
public NavigableSet<E> tailSet(E fromElement, boolean inclusive) {
    return new TreeSet<>(m.tailMap(fromElement, inclusive));
}

public SortedSet<E> tailSet(E fromElement) {
    return tailSet(fromElement, true);
}
```

666. 彩蛋

总的来说，比较简单，相信胖友一会会时间就已经看完了。

关于 TreeSet 的总结，只有一句话：TreeSet 是基于 TreeMap 的 Set 实现类。

文章目录

总访客数 次 && 总访问量 次
回到首页