链滴

TreeMap 的排序及比较器问题

作者：wthfeng
原文链接：https://ld246.com/article/1486614512175
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

   TreeMap默认按键的自然顺序升序进行排序，如果有求要按键的倒序排序，或者按值类型进行排序呢?

   在问题开始之前，让我们先回顾一下有关Map及其排基本的知识点

1. 用的最多的HashMap，不保证映射的顺序，特别是它不保证该顺序恒久不变。
2. LinkedHashMap,维持元素的插入顺序。
3. TreeMap中有一个传入比较器的构造函数， Map中的元素可按此比较器进行排序。

   以上3个知识点，前2个作为复习，最后一个才是本次用的重点。要想改变TreeMap的默认比较次序，我们可以在其构造函数中传入一个自己的比较器。TreMap的比较器构造函数如下：

```
  public TreeMap(Comparator<? super K> comparator)
```

Comaprator排序接口定义如下：

```
public interface Comparator<T> {
   int compare(T o1, T o2);
   ..... //若干方法
}
```

Comparator接口必须实现compare()方法。返回的int值的正负表示两值的大小。本着先易后难原则让我们先实现TreeMap按键倒序排序：

```
package top.wthfeng.hello;

import java.util.Comparator;
import java.util.Map;
import java.util.TreeMap;

public class Map2Test{
    public static void main(String[]args){
        Map<String,String> map = new TreeMap<>(new Comparator<String>(){
            public int compare(String o1,String o2){
                return  o2.compareTo(o1); //用正负表示大小值
            }
        });
        //以上4行可用下面一行lambda表达式代替
        //Map<String,String> map1 = new TreeMap<>((o1,o2)->o2.compareTo(o1));
        map.put("zdef","rfgh");
        map.put("asrg","zfg");
        map.put("rgd","dfgh");
        map.put("cbf","gddf");
        for(Map.Entry<String,String> entry:map.entrySet()){
            System.out.println("key:"+entry.getKey()+",:value:"+entry.getValue());
        }
    }
}
//输出结果（倒序）：
key:zdef,:value:rfgh
key:rgd,:value:dfgh
key:cbf,:value:gddf
```

```
key:asrg,:value:zfg
```

   在TreeMap的构造函数中传入实现了Comparator接的类实例（本例以内部匿名类实现，道理都一样，匿名类更简单，当然java8以后更推荐使用lambda法），该类的唯一方法comparaTo()实现了比较算法。这样TreeMap的倒序排列就解决了。下面我们研究TreeMap的按值排序。

   先想想思路，map的按值排序是没有现成方法的，这就要变换一下想法。在集合工具类Collections中有对集合进行排序的方法，还可传入一个比较器按比器进行排序。方法签名如下：

```
 public static <T> void sort(List<T> list,Comparator<? super T> c);
```

   这也就是说要是一个list的话，就像上面一样给传一个较器，再调用Collections.sort()方法就能解决，可这是map啊，那能不能将map转为list啊？直接看面吧：

```
package top.wthfeng.hello;

import java.util.*;

public class MapTest{
    public static void main(String[]args){
        Map<String,String> map = new TreeMap<>();

        map.put("zdef","rfgh");
        map.put("asrg","zfg");
        map.put("rgd","dfgh");
        map.put("cbf","gddf");
        //将Map转为List
        List<Map.Entry<String,String>> list = new ArrayList<>(map.entrySet());
        Collections.sort(list, new Comparator<Map.Entry<String, String>>() {
            public int compare(Map.Entry<String, String> o1, Map.Entry<String, String> o2) {
                return  o2.getValue().compareTo(o1.getValue());
            }
        }); //重新排序
        //运用lambda表达式
        //Collections.sort(list,((o1, o2) -> o2.getValue().compareTo(o1.getValue())));
        for(Map.Entry<String,String> entry:list){
            System.out.println("key:"+entry.getKey()+",:value:"+entry.getValue());
        }
    }
}
//输出（按值倒序）
key:asrg,:value:zfg
key:zdef,:value:rfgh
key:cbf,:value:gddf
key:rgd,:value:dfgh
```

   OK,TreeMap的按值排序就这样解决了。让我们总结下，以上2个例子用到了Comparator这个接口，而这个接口到底是个怎样的存在呢？

强行对某个对象 collection 进行整体排序 的比较函数。可以使用 Comparator 来控制某些数据结构如有序 set或有序映射）的顺序，或者为那些没有自然顺序的对象 collection 提供排序。

   以上是Java API6上的说明，我又参考了其他资料，大可以认为：是为那些没有排序方法的类自定义一个排序方法的一种手段，由于和原数据类没有耦合，称之为外部比较器。比如说你写了一个苹果类，Apple有weight属性。现要将Apple以weight升序放list中，那么你就可以像上面那样，写个类实现Comparator，在Collections.sort()中实现排序。

```
package top.wthfeng.hello.test;

/**
 * 苹果类
 */
public class Apple {
    /**
     * 重量
     */
    private Integer weight;
    /**
     * 价格
     */
    private Integer price;

    public Integer getPrice() {
        return price;
    }

    public void setPrice(Integer price) {
        this.price = price;
    }

    public Integer getWeight() {
        return weight;
    }

    public void setWeight(Integer weight) {
        this.weight = weight;
    }

    @Override
    public String toString() {   //重写toString()方法，方面输出
         StringBuilder sb = new StringBuilder();
        sb.append("[");
        sb.append("Apple:(weight:");
        sb.append(weight);
        sb.append(",price:");
        sb.append(price);
        sb.append(")]");
        return sb.toString();
    }
}

package top.wthfeng.hello;

import top.wthfeng.hello.test.Apple;

import java.util.*;
```

```
public class Test {  //测试类
    public static void main(String[] args) {

        List<Apple> apples = new ArrayList<>();
        Random random = new Random(12);
        for (int i = 0; i < 10; i++) {  //生成10个苹果，重量随机生成
            Apple apple = new Apple();
            apple.setWeight(random.nextInt(1000));
            apples.add(apple);
        }
        for (Apple apple : apples) { //打印10个苹果的顺序
            System.out.println("apple = " + apple);
        }
        Collections.sort(apples, new Comparator<Apple>() { //排序，传入一个比较器
            @Override
            public int compare(Apple o1, Apple o2) {
                return o1.getWeight().compareTo(o2.getWeight());
            }
        });
        //  Collections.sort(apples,(o1,o2)->o1.getWeight().compareTo(o2.getWeight()));
        for (Apple apple : apples) { //排序后的顺序
            System.out.println(" sort apple = " + apple);
        }
    }
}
//输出
apple = [Apple:(weight:866,price:12)]
apple = [Apple:(weight:556,price:33)]
apple = [Apple:(weight:624,price:11)]
apple = [Apple:(weight:750,price:15)]
apple = [Apple:(weight:596,price:21)]
apple = [Apple:(weight:568,price:22)]
apple = [Apple:(weight:61,price:7)]
apple = [Apple:(weight:695,price:14)]
apple = [Apple:(weight:536,price:31)]
apple = [Apple:(weight:505,price:3)]
 sort apple = [Apple:(weight:61,price:7)]
 sort apple = [Apple:(weight:505,price:3)]
 sort apple = [Apple:(weight:536,price:31)]
 sort apple = [Apple:(weight:556,price:33)]
 sort apple = [Apple:(weight:568,price:22)]
 sort apple = [Apple:(weight:596,price:21)]
 sort apple = [Apple:(weight:624,price:11)]
 sort apple = [Apple:(weight:695,price:14)]
 sort apple = [Apple:(weight:750,price:15)]
 sort apple = [Apple:(weight:866,price:12)]
```

   按weight排序完成。总结一下：我们自定义的类想按个字段排序，可以利用Collections的sort方法传入一个自定义的比较器，这种比较器与被比较的类不生耦合，称为外部比较器。

   那么问题来了，如果我的Apple类默认排序是按价格特殊情况才按重量。总不能每次排序时都要写遍比较器实现吧？这也太麻烦了。不知大家注意到没有在实现Comparator接口中，都有类似下面的句子：

return o2.compareTo(o1);

   那compareTo()方法从哪来的？o1,o2是String类型，ompareTo()正是String实现的Comparable接口的方法。那么Comparable又是什么鬼？

Comparable接口对实现它的每个类的对象进行整体排序，这种排序称为类的自然排序，类的compareTo方法被称为类的比较方法。

   有点眉目了，再看看Comparable的解释，发现java所有值类都实现了Comparable方法。像String、Integer、Byte等等。这些java内置的值类就是根据compare方法比较大小的，尤其重要的是，**若类实现了Comparable接口，它就跟许多泛型算法及依赖该接口的集合比较算法相关。**这就是类的内部排序。

```
package top.wthfeng.hello.test;

/**
 * 苹果类
 */
public class Apple implements Comparable<Apple>{


    /**
     * 重量
     */
    private Integer weight;
    /**
     * 价格
     */
    private Integer price;

    public Integer getPrice() {
        return price;
    }

    public void setPrice(Integer price) {
        this.price = price;
    }

    public Integer getWeight() {
        return weight;
    }

    public void setWeight(Integer weight) {
        this.weight = weight;
    }

    @Override
    public String toString() {   //重写toString()方法，方面输出
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        sb.append("Apple:(weight:");
        sb.append(weight);
        sb.append(",price:");
        sb.append(price);
```

```
        sb.append(")]");
        return sb.toString();
    }

    @Override
    public int compareTo(Apple o) {  //实现内部排序
        return this.price.compareTo(o.getPrice());
    }
}

package top.wthfeng.hello;

import top.wthfeng.hello.test.Apple;

import java.util.*;

public class Test {
    public static void main(String[] args) {

        List<Apple> apples = new ArrayList<>();
        Random random = new Random(12);
        for (int i = 0; i < 10; i++) {  //生成10个苹果，重量随机生成
            Apple apple = new Apple();
            apple.setWeight(random.nextInt(1000));
            apple.setPrice(random.nextInt(50));
            apples.add(apple);
        }
        for (Apple apple : apples) { //打印10个苹果的顺序
            System.out.println("apple = " + apple);
        }
       Collections.sort(apples);
        //  Collections.sort(apples,(o1,o2)->o1.getWeight().compareTo(o2.getWeight()));
        for (Apple apple : apples) {
            System.out.println(" sort apple = " + apple);
        }
    }
}

//输出
apple = [Apple:(weight:866,price:12)]
apple = [Apple:(weight:556,price:33)]
apple = [Apple:(weight:624,price:11)]
apple = [Apple:(weight:750,price:15)]
apple = [Apple:(weight:596,price:21)]
apple = [Apple:(weight:568,price:22)]
apple = [Apple:(weight:61,price:7)]
apple = [Apple:(weight:695,price:14)]
apple = [Apple:(weight:536,price:31)]
apple = [Apple:(weight:505,price:3)]
 sort apple = [Apple:(weight:505,price:3)]
 sort apple = [Apple:(weight:61,price:7)]
 sort apple = [Apple:(weight:624,price:11)]
 sort apple = [Apple:(weight:866,price:12)]
 sort apple = [Apple:(weight:695,price:14)]
```

```
sort apple = [Apple:(weight:750,price:15)]
sort apple = [Apple:(weight:596,price:21)]
sort apple = [Apple:(weight:568,price:22)]
sort apple = [Apple:(weight:536,price:31)]
sort apple = [Apple:(weight:556,price:33)]
```