

skywang12345

导航

- 博客园
- 首页
- 新随笔
- 联系
- 订阅 XML
- 管理

2019年8月						
日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

统计

- 随笔 - 278
- 文章 - 0
- 评论 - 1386
- 引用 - 0

搜索

找找看

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

随笔分类 (275)

- Android(7)
- Android NDK编程(9)
- Android 系统层(5)
- Android 应用层(46)
- Computer Culture(2)
- Java(111)
- Linux/Ubuntu(5)
- UML(5)
- Windows(1)
- 设计模式(1)
- 数据结构_算法(79)
- 索引(4)

最新评论

Java 集合系列12之 TreeMap详细介绍(源码解析)和使用示例

概要

这一章，我们对TreeMap进行学习。
我们先对TreeMap有个整体认识，然后再学习它的源码，最后再通过实例来学会使用TreeMap。内容包括：
第1部分 TreeMap介绍
第2部分 TreeMap数据结构
第3部分 TreeMap源码解析(基于JDK1.6.0_45)
第4部分 TreeMap遍历方式
第5部分 TreeMap示例

转载请注明出处：
<http://www.cnblogs.com/skywang12345/admin/EditPosts.aspx?postid=3310928>

第1部分 TreeMap介绍

TreeMap 简介

TreeMap 是一个**有序的key-value集合**，它是通过**红黑树**实现的。
TreeMap **继承于AbstractMap**，所以它是一个Map，即一个key-value集合。
TreeMap 实现了NavigableMap接口，意味着它**支持一系列的导航方法**。比如返回有序的key集合。
TreeMap 实现了Cloneable接口，意味着**它能被克隆**。
TreeMap 实现了java.io.Serializable接口，意味着**它支持序列化**。
TreeMap基于**红黑树 (Red-Black tree)** 实现。该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的**Comparator 进行排序**，具体取决于使用的构造方法。
TreeMap的基本操作 containsKey、get、put 和 remove 的时间复杂度是 log(n)。
另外，TreeMap是**非同步的**。它的iterator 方法返回的**迭代器是fail-fast**的。

1. Re:Java多线程系列--“基础篇”08之join()
我看Java编程思想的例子里没有父子线程关系也可以使用join()啊///
concurrency/Joining.javapackage
concurrency; /* Added by
Eclip.....
--血手||人屠
2. Re:Java 集合系列13之WeakHashMap详细介绍(源码解析)和使用示例
private void expungeStaleEntries()
{ Entry e; while ((e = (Entry)
queue.poll()) !=
--墨染-
3. Re:Java 集合系列04之 fail-fast总结(通过ArrayList来说明fail-fast的原理、解决办法)
@甜甜咿呀咿呀哟更新操作源码里没有改变modcount值，所以你说的对...
--墨染-
4. Re:[转载] 散列表(Hash Table) 从理论到实用 (下)
很厉害，前两篇倒是看懂了 最后一篇看不下去了，大概是耐心都被前面的耗尽了 收获不少 下次重新看最后一篇
--奇怪的程序猿
5. Re:Java多线程系列目录(共43篇)
没看完，感觉很厉害，都整理成这个样子了
--南宫慕容007

阅读排行榜

1. 红黑树(一)之 原理和算法详细介绍(396645)
2. Java 集合系列10之 HashMap详细介绍(源码解析)和使用示例(286463)
3. Java 集合系列03之 ArrayList详细介绍(源码解析)和使用示例(161483)
4. Java 集合系列12之 TreeMap详细介绍(源码解析)和使用示例(158854)
5. 图的遍历之 深度优先搜索和广度优先搜索(158690)

TreeMap的构造函数

```
// 默认构造函数。使用该构造函数，TreeMap中的元素按照自然排序进行排列。  
TreeMap()  
  
// 创建的TreeMap包含Map  
TreeMap(Map<? extends K, ? extends V> copyFrom)  
  
// 指定Tree的比较器  
TreeMap(Comparator<? super K> comparator)  
  
// 创建的TreeSet包含copyFrom  
TreeMap(SortedMap<K, ? extends V> copyFrom)
```

TreeMap的API

Entry<K, V>	ceilingEntry(K key)
K	ceilingKey(K key)
void	clear()
Object	clone()
Comparator<? super K>	comparator()
boolean	containsKey(Object key)
NavigableSet<K>	descendingKeySet()
NavigableMap<K, V>	descendingMap()
Set<Entry<K, V>>	entrySet()
Entry<K, V>	firstEntry()
K	firstKey()
Entry<K, V>	floorEntry(K key)
K	floorKey(K key)
V	get(Object key)
NavigableMap<K, V>	headMap(K to, boolean inclusive)
SortedMap<K, V>	headMap(K toExclusive)
Entry<K, V>	higherEntry(K key)
K	higherKey(K key)
boolean	isEmpty()
Set<K>	keySet()
Entry<K, V>	lastEntry()
K	lastKey()
Entry<K, V>	lowerEntry(K key)
K	lowerKey(K key)
NavigableSet<K>	navigableKeySet()
Entry<K, V>	pollFirstEntry()
Entry<K, V>	pollLastEntry()
V	put(K key, V value)
V	remove(Object key)
int	size()
SortedMap<K, V>	subMap(K fromInclusive, K toExclusive)
NavigableMap<K, V>	subMap(K from, boolean fromInclusive, K to, boolean toInclusive)

```
NavigableMap<K, V>          tailMap(K from, boolean inclusive)
SortedMap<K, V>             tailMap(K fromInclusive)
```



第2部分 TreeMap数据结构

TreeMap的继承关系

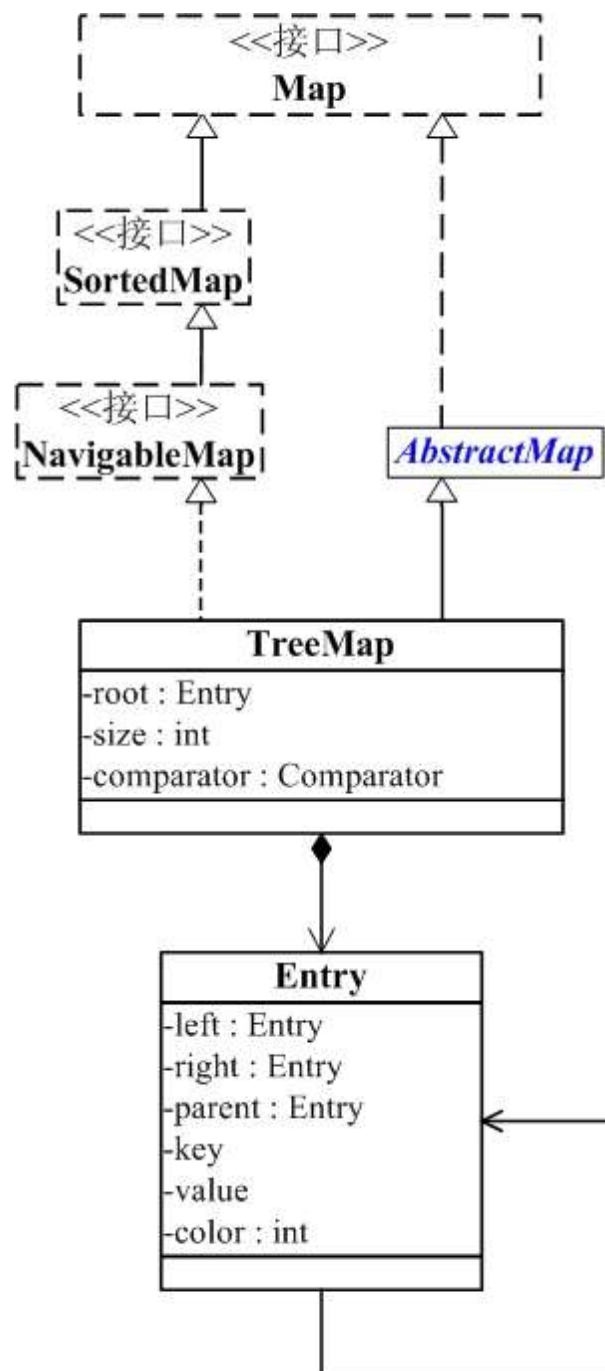


```
java.lang.Object
├── java.util.AbstractMap<K, V>
│   └── java.util.TreeMap<K, V>

public class TreeMap<K,V>
    extends AbstractMap<K,V>
    implements NavigableMap<K,V>, Cloneable,
java.io.Serializable {}
```



TreeMap与Map关系如下图:



从图中可以看出：

(01) `TreeMap`实现继承于`AbstractMap`，并且实现了`NavigableMap`接口。

(02) `TreeMap`的本质是R-B Tree(红黑树)，它包含几个重要的成员变量：`root`、`size`、`comparator`。

`root` 是红黑数的根节点。它是`Entry`类型，`Entry`是红黑数的节点，它包含了红黑数的6个基本组成成分：`key`(键)、`value`(值)、`left`(左孩子)、`right`(右孩子)、`parent`(父节点)、`color`(颜色)。 `Entry`节点根据`key`进行排序，`Entry`节点包含的内容为`value`。

红黑数排序时，根据`Entry`中的`key`进行排序；`Entry`中的`key`比较大小是根据比较器`comparator`来进行判断的。

`size`是红黑数中节点的个数。

关于红黑数的具体算法，请参考["红黑树\(一\) 原理和算法详细介绍"](#)。

第3部分 TreeMap源码解析（基于JDK1.6.0_45）

为了更了解TreeMap的原理，下面对TreeMap源码代码作出分析。我们先给出源码内容，后面再对源码进行详细说明，当然，源码内容中也包含了详细的代码注释。读者阅读的时候，建议先看后面的说明，先建立一个整体印象；之后再阅读源码。

[View Code](#)

说明：

在详细介绍TreeMap的代码之前，我们先建立一个整体概念。TreeMap是通过红黑树实现的，TreeMap存储的是key-value键值对，TreeMap的排序是基于对key的排序。TreeMap提供了操作“key”、“key-value”、“value”等方法，也提供了对TreeMap这颗树进行整体操作的方法，如获取子树、反向树。

后面的解说内容分为几部分，

首先，介绍TreeMap的核心，即红黑树相关部分；

然后，介绍TreeMap的主要函数；

再次，介绍TreeMap实现的几个接口；

最后，补充介绍TreeMap的其它内容。

TreeMap本质上是一颗红黑树。要彻底理解TreeMap，建议读者先理解红黑树。关于红黑树的原理，可以参考：[红黑树](#)

[\(一\) 原理和算法详细介绍](#)

第3.1部分 TreeMap的红黑树相关内容

TreeMap中于红黑树相关的主要函数有：

1 数据结构

1.1 红黑树的节点颜色--红色

```
private static final boolean RED = false;
```

1.2 红黑树的节点颜色--黑色

```
private static final boolean BLACK = true;
```

1.3 “红黑树的节点”对应的类。

```
static final class Entry<K,V> implements Map.Entry<K,V> { ...
}
```

Entry包含了6个部分内容：key(键)、value(值)、left(左孩子)、right(右孩子)、parent(父节点)、color(颜色)
Entry节点根据key进行排序，Entry节点包含的内容为value。

2 相关操作

2.1 左旋

```
private void rotateLeft(Entry<K,V> p) { ... }
```

2.2 右旋

```
private void rotateRight(Entry<K,V> p) { ... }
```

2.3 插入操作

```
public V put(K key, V value) { ... }
```

2.4 插入修正操作

红黑树执行插入操作之后，要执行“插入修正操作”。

目的是：**保红黑树在进行插入节点之后，仍然是一颗红黑树**

```
private void fixAfterInsertion(Entry<K,V> x) { ... }
```

2.5 删除操作

```
private void deleteEntry(Entry<K,V> p) { ... }
```

2.6 删除修正操作

红黑树执行删除之后，要执行“删除修正操作”。

目的是保证：**红黑树删除节点之后，仍然是一颗红黑树**

```
private void fixAfterDeletion(Entry<K,V> x) { ... }
```

关于红黑树部分，这里主要是指出了TreeMap中那些是红黑树的主要相关内容。具体的红黑树相关操作API，这里没有详细说明，因为它们仅仅只是将算法翻译成代码。读者可以参考[“红黑树\(一\) 原理和算法详细介绍”](#)进行了了解。

第3.2部分 TreeMap的构造函数

1 默认构造函数

使用默认构造函数构造TreeMap时，使用java的默认的比较器比较Key的大小，从而对TreeMap进行排序。

```
public TreeMap() {
    comparator = null;
```

```
}
```

2 带比较器的构造函数

```
public TreeMap(Comparator<? super K> comparator) {  
    this.comparator = comparator;  
}
```

3 带Map的构造函数，Map会成为TreeMap的子集

```
public TreeMap(Map<? extends K, ? extends V> m) {  
    comparator = null;  
    putAll(m);  
}
```

该构造函数会调用putAll()将m中的所有元素添加到TreeMap中。putAll()源码如下：

```
public void putAll(Map<? extends K, ? extends V> m) {  
    for (Map.Entry<? extends K, ? extends V> e :  
m.entrySet())  
        put(e.getKey(), e.getValue());  
}
```

从中，我们可以看出putAll()就是将m中的key-value逐个的添加到TreeMap中。

4 带SortedMap的构造函数，SortedMap会成为TreeMap的子集



```
public TreeMap(SortedMap<K, ? extends V> m) {  
    comparator = m.comparator();  
    try {  
        buildFromSorted(m.size(), m.entrySet().iterator(),  
null, null);  
    } catch (java.io.IOException cannotHappen) {  
    } catch (ClassNotFoundException cannotHappen) {  
    }  
}
```



该构造函数不同于上一个构造函数，在上一个构造函数中传入的参数是Map，Map不是有序的，所以要逐个添加。

而该构造函数的参数是SortedMap是一个有序的Map，我们通过buildFromSorted()来创建对应的Map。

buildFromSorted涉及到的代码如下：

[View Code](#)

要理解buildFromSorted，重点说明以下几点：

第一, buildFromSorted是**通过递归将SortedMap中的元素逐个关联**。


第二, buildFromSorted返回**middle节点(中间节点)作为root**。

第三, buildFromSorted**添加到红黑树中时, 只将level == redLevel的节点设为红色**。第level级节点, 实际上是buildFromSorted转换成红黑树后的最底端(假设根节点在最上方)的节点; 只将红黑树最底端的阶段着色为红色, 其余都是黑色。

第3.3部分 TreeMap的Entry相关函数


TreeMap的 **firstEntry()**、**lastEntry()**、**lowerEntry()**、**higherEntry()**、**floorEntry()**、**ceilingEntry()**、**pollFirstEntry()**、**pollLastEntry()** 原理都是类似的; 下面以firstEntry()来进行详细说明

我们先看看firstEntry()和getFirstEntry()的代码:



```
public Map.Entry<K,V> firstEntry() {
    return exportEntry(getFirstEntry());
}

final Entry<K,V> getFirstEntry() {
    Entry<K,V> p = root;
    if (p != null)
        while (p.left != null)
            p = p.left;
    return p;
}
```



从中, 我们可以看出 firstEntry() 和 getFirstEntry() 都是用于**获取第一个节点**。

但是, firstEntry() 是**对外接口**; getFirstEntry() 是**内部接口**。而且, firstEntry() 是通过 getFirstEntry() 来实现的。那为什么外界不能直接调用 getFirstEntry(), 而需要多此一举的调用 firstEntry() 呢?

先告诉大家原因, 再进行详细说明。这么做的目的是: **防止用户修改返回的Entry**。getFirstEntry()返回的Entry是可以被修改的, 但是经过firstEntry()返回的Entry不能被修改, 只可以读取Entry的key值和value值。下面我们看看到底是如何实现的。
(01) getFirstEntry()返回的是Entry节点, 而Entry是红黑树的节点, 它的源码如下:




```
// 返回“红黑树的第一个节点”
final Entry<K,V> getFirstEntry() {
    Entry<K,V> p = root;
    if (p != null)
        while (p.left != null)
            p = p.left;
    return p;
}
```



从中，我们可以调用Entry的getKey()、getValue()来获取key和value值，以及调用setValue()来修改value的值。

(02) firstEntry()返回的是exportEntry(getFirstEntry())。下面我们看看exportEntry()干了些什么？

```
static <K,V> Map.Entry<K,V> exportEntry(TreeMap.Entry<K,V> e)
{
    return e == null? null :
        new AbstractMap.SimpleImmutableEntry<K,V>(e);
}
```

实际上，exportEntry() 是新建一个AbstractMap.SimpleImmutableEntry类型的对象，并返回。

SimpleImmutableEntry的实现在AbstractMap.java中，下面我们看看AbstractMap.SimpleImmutableEntry是如何实现的，代码如下：

[View Code](#)

从中，我们可以看出SimpleImmutableEntry实际上是简化的key-value节点。

它只提供了getKey()、getValue()方法类获取节点的值；但不能修改value的值，因为调用 setValue() 会抛出异常UnsupportedOperationException();

再回到我们之前的问题：那为什么外界不能直接调用getFirstEntry()，而需要多此一举的调用 firstEntry() 呢？现在我们清晰的了解到：

(01) firstEntry()是对外接口，而getFirstEntry()是内部接口。

(02) 对firstEntry()返回的Entry对象只能进行getKey()、getValue()等读取操作；而对getFirstEntry()返回的对象除了可以进行读取操作之后，还可以通过setValue()修改值。

第3.4部分 TreeMap的key相关函数

TreeMap的**firstKey()**、**lastKey()**、**lowerKey()**、**higherKey()**、**floorKey()**、**ceilingKey()**原理都是类似的；下面以ceilingKey()来进行详细说明

ceilingKey(K key)的作用是“返回大于/等于key的最小的键值对所对应的KEY，没有的话返回null”，它的代码如下：

```
public K ceilingKey(K key) {  
    return keyOrNull(getCeilingEntry(key));  
}
```

ceilingKey()是通过getCeilingEntry()实现的。keyOrNull()的代码很简单，它是获取节点的key，没有的话，返回null。

```
static <K,V> K keyOrNull(TreeMap.Entry<K,V> e) {  
    return e == null? null : e.key;  
}
```

getCeilingEntry(K key)的作用是“获取TreeMap中大于/等于key的最小的节点，若不存在(即TreeMap中所有节点的键都比key大)，就返回null”。它的实现代码如下：

[View Code](#)

第3.5部分 TreeMap的values()函数

values() 返回“**TreeMap中值的集合**”

values()的实现代码如下：

```
public Collection<V> values() {  
    Collection<V> vs = values;  
    return (vs != null) ? vs : (values = new Values());  
}
```

说明：从中，我们可以发现values()是通过 new Values() 来实现“返回TreeMap中值的集合”。

那么Values()是如何实现的呢？ 没错！由于返回的是值的集合，那么Values()肯定返回一个集合；而Values()正好是集合类Value的构造函数。Values继承于AbstractCollection，它的代码如下：

[View Code](#)

说明：从中，我们可以知道Values类就是一个集合。而AbstractCollection 实现了除 size() 和 iterator() 之外的其它函数，因此只需要在Values类中实现这两个函数即可。size() 的实现非常简单，Values集合中元素的个数=该TreeMap的元素个数。(TreeMap每一个元素都有一个值嘛！)

iterator() 则返回一个迭代器，用于遍历Values。下面，我们一起可以看看iterator()的实现：

```
public Iterator<V> iterator() {  
    return new ValueIterator(getFirstEntry());  
}
```

说明： iterator() 是通过ValueIterator() 返回迭代器的，ValueIterator是一个类。代码如下：

```
final class ValueIterator extends PrivateEntryIterator<V> {  
    ValueIterator(Entry<K,V> first) {  
        super(first);  
    }  
    public V next() {  
        return nextEntry().value;  
    }  
}
```

说明：ValueIterator的代码很简单，它的主要实现应该在它的父类PrivateEntryIterator中。下面我们一起来看看PrivateEntryIterator的代码：

[View Code](#)

说明：PrivateEntryIterator是一个抽象类，它的实现很简单，只实现了Iterator的remove()和hasNext()接口，没有实现next()接口。

而我们在ValueIterator中已经实现的next()接口。

至此，我们就了解了iterator()的完整实现了。

第3.6部分 TreeMap的entrySet()函数

entrySet() 返回“键值对集合”。顾名思义，它返回的是一个集合，集合的元素是“键值对”。

下面，我们看看它是如何实现的？entrySet() 的实现代码如下：

```
public Set<Map.Entry<K,V>> entrySet() {  
    EntrySet es = entrySet;  
    return (es != null) ? es : (entrySet = new EntrySet());  
}
```

说明：entrySet()返回的是一个EntrySet对象。

下面我们看看EntrySet的代码：

[View Code](#)

说明：

EntrySet是“**TreeMap的所有键值对组成的集合**”，而且它单位是单个“键值对”。

EntrySet是一个集合，它继承于AbstractSet。而AbstractSet实现了除size() 和 iterator() 之外的其它函数，因此，我们重点了解一下EntrySet的size() 和 iterator() 函数

size() 的实现非常简单，AbstractSet集合中元素的个数=该TreeMap的元素个数。

iterator() 则返回一个迭代器，用于遍历AbstractSet。从上面的源码中，我们可以发现**iterator() 是通过EntryIterator实现的**；下面我们看看EntryIterator的源码：



```
final class EntryIterator extends
PrivateEntryIterator<Map.Entry<K,V>> {
    EntryIterator(Entry<K,V> first) {
        super(first);
    }
    public Map.Entry<K,V> next() {
        return nextEntry();
    }
}
```



说明：和Values类一样，EntryIterator也继承于PrivateEntryIterator类。

第3.7部分 TreeMap实现的Cloneable接口

TreeMap实现了Cloneable接口，即**实现了clone()方法**。clone()方法的作用很简单，就是克隆一个TreeMap对象并返回。

[View Code](#)

第3.8部分 TreeMap实现的Serializable接口

TreeMap实现java.io.Serializable，分别实现了串行读取、写入功能。

串行写入函数是writeObject()，它的作用是将**TreeMap的“容量、所有的Entry”都写入到输出流中**。

而串行读取函数是readObject()，它的作用是将**TreeMap的“容量、所有的Entry”依次读出**。

readObject() 和 writeObject() 正好是一对，通过它们，我能实现TreeMap的串行传输。

[View Code](#)

说到这里，就顺便说一下“关键字transient”的作用

transient是Java语言的关键字，它被用来表示一个域不是该对象串行化的一部分。

Java的serialization提供了一种持久化对象实例的机制。当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

第3.9部分 TreeMap实现的NavigableMap接口

firstKey()、lastKey()、lowerKey()、higherKey()、ceilingKey()、floorKey();
firstEntry()、lastEntry()、lowerEntry()、higherEntry()、floorEntry()、ceilingEntry()、pollFirstEntry()、pollLastEntry();

上面已经讲解过这些API了，下面对其它的API进行说明。

1 反向TreeMap

descendingMap() 的作用是返回当前TreeMap的反向的TreeMap。所谓反向，就是排序顺序和原始的顺序相反。

我们已经知道TreeMap是一颗红黑树，而红黑树是有序的。TreeMap的排序方式是通过比较器，在创建TreeMap的时候，若指定了比较器，则使用该比较器；否则，就使用Java的默认比较器。

而获取TreeMap的反向TreeMap的原理就是将比较器反向即可！

理解了descendingMap()的反向原理之后，再讲解一下descendingMap()的代码。



```
// 获取TreeMap的降序Map
public NavigableMap<K, V> descendingMap() {
    NavigableMap<K, V> km = descendingMap;
    return (km != null) ? km :
        (descendingMap = new DescendingSubMap(this,
                                                true, null,
true,
                                                true, null,
true));
}
```



从中，我们看出descendingMap()实际上是返回DescendingSubMap类的对象。下面，看看DescendingSubMap的源码：

[View Code](#)

从中，我们看出DescendingSubMap是降序的SubMap，它的实现机制是将“SubMap的比较器反转”。

它继承于NavigableSubMap。而NavigableSubMap是一个继承于AbstractMap的抽象类；它包括2个子类——“(升序)AscendingSubMap”和“(降序)DescendingSubMap”。NavigableSubMap为它的两个子类实现了许多公共API。下面看看NavigableSubMap的源码。

[View Code](#)

NavigableSubMap源码很多，但不难理解；读者可以通过源码和注释进行理解。

其实，读完NavigableSubMap的源码后，我们可以得出它的核心思想是：它是一个抽象集合类，为2个子类——“(升序)AscendingSubMap”和“(降序)DescendingSubMap”而服务；因为NavigableSubMap实现了许多公共API。它的最终目的是实现下面的一系列函数：



```
headMap(K toKey, boolean inclusive)
headMap(K toKey)
subMap(K fromKey, K toKey)
subMap(K fromKey, boolean fromInclusive, K toKey, boolean
toInclusive)
tailMap(K fromKey)
tailMap(K fromKey, boolean inclusive)
navigableKeySet()
descendingKeySet()
```



第3.10部分 TreeMap其它函数

1 顺序遍历和逆序遍历

TreeMap的顺序遍历和逆序遍历原理非常简单。

由于TreeMap中的元素是**从小到大的顺序排列的**。因此，顺序遍历，就是从第一个元素开始，逐个向后遍历；而倒序遍历则恰恰相反，它是从最后一个元素开始，逐个往前遍历。

我们可以通过 keyIterator() 和 descendingKeyIterator()来说明！


keyIterator()的作用是**返回顺序的KEY的集合**,
descendingKeyIterator()的作用是**返回逆序的KEY的集合**。

keyIterator() 的代码如下:


```
Iterator<K> keyIterator() {  
    return new KeyIterator(getFirstEntry());  
}
```

说明: 从中我们可以看出**keyIterator()** 是返回以“第一个节点(getFirstEntry)”为其实元素的迭代器。

KeyIterator的代码如下:



```
final class KeyIterator extends PrivateEntryIterator<K> {  
    KeyIterator(Entry<K,V> first) {  
        super(first);  
    }  
    public K next() {  
        return nextEntry().key;  
    }  
}
```




说明: **KeyIterator**继承于**PrivateEntryIterator**。当我们通过**next()**不断获取下一个元素的时候, 就是执行的顺序遍历了。

descendingKeyIterator()的代码如下:


```
Iterator<K> descendingKeyIterator() {  
    return new DescendingKeyIterator(getLastEntry());  
}
```

说明: 从中我们可以看出**descendingKeyIterator()** 是返回以“最后一个节点(getLastEntry)”为其实元素的迭代器。

再看看**DescendingKeyIterator**的代码:



```
final class DescendingKeyIterator extends  
PrivateEntryIterator<K> {  
    DescendingKeyIterator(Entry<K,V> first) {  
        super(first);  
    }  
    public K next() {  
        return prevEntry().key;  
    }  
}
```



说明: **DescendingKeyIterator**继承于**PrivateEntryIterator**。当我们通过**next()**不断获取下一个元素

的时候，实际上调用的是prevEntry()获取的上一个节点，这样它实际上执行的是逆序遍历了。

至此，TreeMap的相关内容就全部介绍完毕了。若有错误或纰漏的地方，欢迎指正！

第4部分 TreeMap遍历方式

4.1 遍历TreeMap的键值对

第一步：根据entrySet()获取TreeMap的“键值对”的Set集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。



```
// 假设map是TreeMap对象
// map中的key是String类型, value是Integer类型
Integer integ = null;
Iterator iter = map.entrySet().iterator();
while(iter.hasNext()) {
    Map.Entry entry = (Map.Entry)iter.next();
    // 获取key
    key = (String)entry.getKey();
    // 获取value
    integ = (Integer)entry.getValue();
}
```



4.2 遍历TreeMap的键

第一步：根据keySet()获取TreeMap的“键”的Set集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。



```
// 假设map是TreeMap对象
// map中的key是String类型, value是Integer类型
String key = null;
Integer integ = null;
Iterator iter = map.keySet().iterator();
while (iter.hasNext()) {
    // 获取key
    key = (String)iter.next();
    // 根据key, 获取value
    integ = (Integer)map.get(key);
}
```



4.3 遍历TreeMap的值

第一步：根据value()获取TreeMap的“值”的集合。

第二步：通过Iterator迭代器遍历“第一步”得到的集合。



```
// 假设map是TreeMap对象
// map中的key是String类型, value是Integer类型
Integer value = null;
Collection c = map.values();
Iterator iter= c.iterator();
while (iter.hasNext()) {
    value = (Integer)iter.next();
}
```



TreeMap遍历测试程序如下:

[View Code](#)

第5部分 TreeMap 示例

下面通过实例来学习如何使用TreeMap

[View Code](#)

运行结果:



```
{one=8, three=4, two=2}
next : one - 8
next : three - 4
next : two - 2
size: 3
contains key two : true
contains key five : false
contains value 0 : false
tmap:{one=8, two=2}
tmap is empty
```



生活的悲欢离合永远在地平线以外，而眺望

是一种青春的姿态...

PS. 文章是笔者分享的学习笔记，若你觉得可以、还行、过得去、甚至不太差的话，可以“推荐”一下的哦。就此谢过！

分类: [Java](#)

标签: [总结](#), [Java](#), [Set](#), [Map](#), [collection](#), [list](#), [集合](#), [系列](#), [iterator](#), [框架](#), [Treemap](#)

好文要顶

关注我

收藏该文

如果天空不死

关注 - 9

粉丝 - 2839

+加关注

41

1

« 上一篇: [Java 集合系列11之 Hashtable详细介绍\(源码解析\)和使用示例](#)
» 下一篇: [Java 集合系列14之 Map总结\(HashMap, Hashtable, TreeMap, WeakHashMap等使用场景\)](#)
posted on 2013-09-23 09:16 [如果天空不死](#) 阅读(158855) 评论(11) [编辑](#) [收藏](#)

Comments

- #1楼

[memristor](#)

Posted @ 2014-07-22 23:05

楼主真是牛逼!!!

支持(1) 反对(0)
- #2楼

[zhidan](#)

Posted @ 2014-10-18 23:51

read it later

支持(1) 反对(0)
- #3楼

[蓝枫居士](#)

Posted @ 2014-10-23 20:43

楼主牛逼!

支持(0) 反对(0)
- #4楼

[一直乱跑的熊](#)

Posted @ 2016-10-03 11:14

牛逼

支持(0) 反对(0)
- #5楼

[and1990](#)

Posted @ 2016-10-20 12:07

牛逼

支持(0) 反对(0)
- #6楼

[BiuBiuBong](#)

Posted @ 2016-12-18 17:05

两千行的源码
我没时间看完 先放放

支持(0) 反对(0)
- #7楼

[lalmeme](#)

Posted @ 2018-01-11 10:29

受教了

[支持\(0\)](#) [反对\(0\)](#)**#8楼**

funnyZpC

Posted @ 2018-01-26 14:07

请问题主，您的博客皮肤是自己做的，还是cnblogs自有皮肤？

[支持\(0\)](#) [反对\(0\)](#)**#9楼**

凜凜

Posted @ 2018-03-17 16:43

楼主。

若getCeilingEntry能走到这一步，那么，它之前“已经遍历过的节点的key”不应该是都< key么

另：

代码里的注释，楼主用的是百度翻译还是谷歌翻译，保护。。。。

[支持\(0\)](#) [反对\(0\)](#)**#10楼**

轻抚、两袖风尘

Posted @ 2018-09-10 10:47

受教了，真厉害

[支持\(0\)](#) [反对\(0\)](#)**#11楼**

hk_zz

Posted @ 2019-05-13 13:53

@ 凜凜

确实，感觉代码里的注释很多都是错的

[支持\(0\)](#) [反对\(0\)](#)[刷新评论](#) [刷新页面](#) [返回顶部](#)**注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。****【推荐】** [超50万C++/C#源码：大型实时仿真组态图形源码](#)**【推荐】** [程序员问答平台，解决您开发中遇到的技术难题](#)**相关博文：**

- [Java 集合系列11之 Hashtable详细介绍\(源码解析\)和使用示例](#)
- [Java 集合系列12之 TreeMap详细介绍\(源码解析\)和使用示例](#)
- [Java 集合系列12之 TreeMap详细介绍\(源码解析\)和使用示例](#)
- [Java 集合系列12之 TreeMap详细介绍\(源码解析\)和使用示例](#)
- [JavaTreeMap详细介绍和使用示例](#)

最新新闻：

- [科学家用千年隼命名新化石物种](#)
 - [无辣不欢其实是种“自虐”](#)
 - [视觉中国上半年财报出炉：“版权门”影响颇大](#)
 - [IBM发布新区块链网络 高效管理全球供应链](#)
 - [全球图像传感器市场：索尼独占过半份额，这家国产厂商排名第三！](#)
- » [更多新闻...](#)