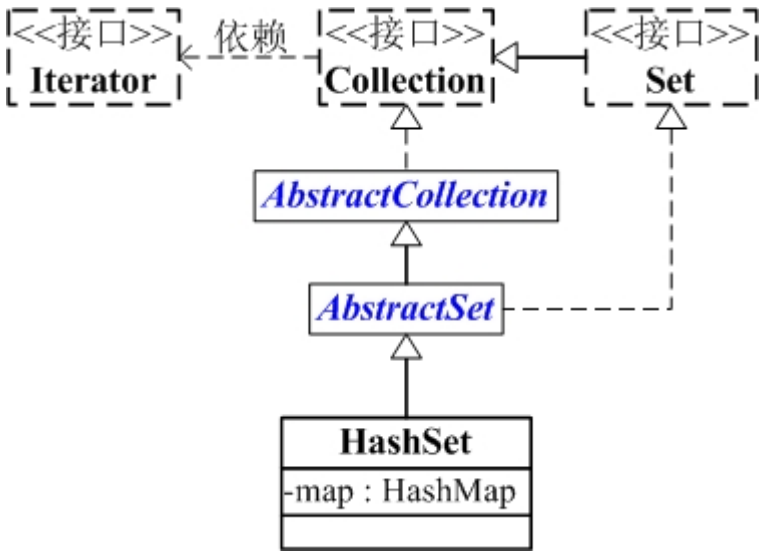
Set实现类

1. HashSet



- `HashSet` 是一个**没有重复元素的集合。**
- 它是由`HashMap`实现的，**不保证元素的顺序**，而且`HashSet`**允许使用** `null` **元素。**
- `HashSet`可以通过迭代器`Iterator`进行遍历。
- `HashSet`内部封装了一个`HashMap`对象，因此它的数据结构和`HashMap`一样，其中`HashMap`的key为要存储的元素，value为一个`Object`。
- 对于`HashSet`中持有的对象，请注意正确重写其`equals`和`hashCode`方法，以保证放入的对象的唯一性。

1.1. 构造函数

```
//构造一个新的空 set，其底层 HashMap 实例的默认初始容量是 16，加载因子是 0.75。
public HashSet() {
        this.map = new HashMap();
}

//构造一个包含指定 Collection 中的元素的新 set。
public HashSet(Set<? extends E> c) {
    map = new HashMap<E,Object>(Math.max((int) (c.size()/.75f) + 1, 16));

    addAll(c);
}

//构造一个新的空 set，其底层 HashMap 实例具有指定的初始容量和默认的加载因子（0.75）。
public HashSet(int initialCapacity, float loadFactor) {
    map = new HashMap<E,Object>(initialCapacity, loadFactor);
}
```

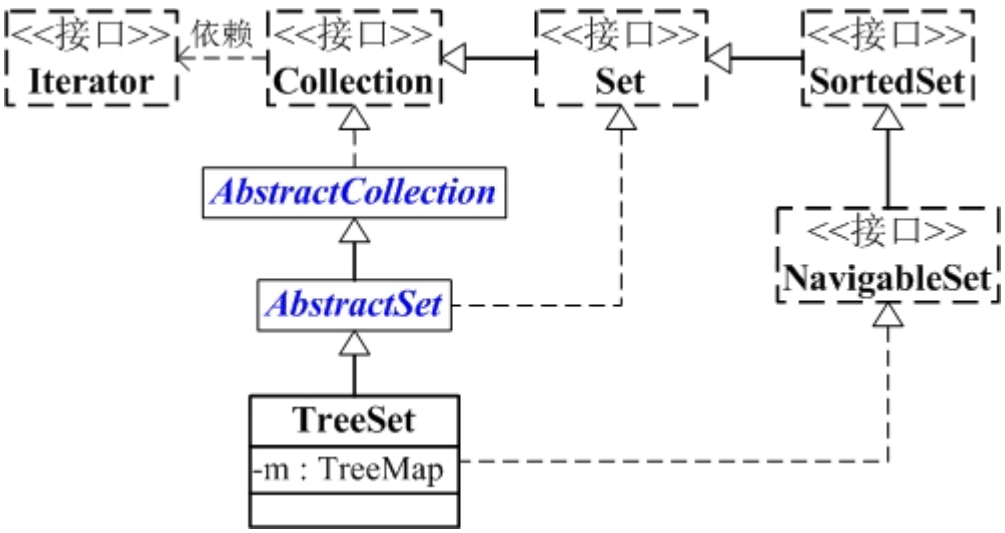
1.2. HashSet主要方法

`HashSet`主要实现了Set接口的方法

`HashSet`中的元素是无序的，不支持根据索引获取元素的方法。

`HashSet`基于`HashMap`实现的，因此其查询，更新时间复杂度为`O(logN)`。

2. TreeSet



- `TreeSet`继承于`AbstractSet`，所以它是一个`Set`容器，具有`Set`的属性和方法，且仅允许有一个空值`null`。

- `TreeSet`实现了`SortedSet`接口，它是有序容器。
- `TreeSet` 实现了`NavigableSet`接口，意味着它支持一系列的导航方法。比如查找与指定目标最匹配项。
- `TreeSet` 实现了`Cloneable`接口，意味着它能被克隆。
- `TreeSet` 实现了`java.io.Serializable`接口，意味着它支持序列化。
- `TreeSet` 封装了`TreeMap`对象，因此要求需要被持有对象实现`Comparable`接口（或提供`Comparator`）并且要求同等性比较时唯一。

2.1. 构造方法

```
// 默认构造函数。使用该构造函数，TreeSet中的元素按照自然排序进行排列。
public TreeSet() {
         this(new TreeMap<E,Object>());
    }

// 创建的TreeSet包含collection
   public TreeSet(Collection<? extends E> c) {
         this();
         addAll(c);
    }

// 指定TreeSet的比较器
    public TreeSet(Comparator<? super E> comparator) {
         this(new TreeMap<>(comparator));
    }


// 创建的TreeSet包含SortedSet
    public TreeSet(SortedSet<E> s) {
         this(s.comparator());
         addAll(s);
    }
```

2.2. TreeSet主要方法

`TreeSet`实现了Set接口主要方法>

`TreeSet`实现了SortedSet接口主要方法

`TreeSet`实现了NavigableSet接口主要方法

3. Set的遍历方式

3.1. 迭代器遍历

```
for(Iterator iter = set.iterator(); iter.hasNext(); ) {
    iter.next();
}
```

3.2. foreach遍历

```
Set<String>set;
for (String str : set) {
      System.out.println(str);
}
```

3.3. TreeSet逆向遍历

`TreeSet`提供`Iterator<E> descendingIterator()`返回在此 Set 的元素上的逆序迭代器。

```
 for(Iterator iter = set.descendingIterator(); iter.hasNext(); ){
        iter.next();
 }
```

4. Set接口主要方法

```
boolean add(E e)                              //向Set末尾中添加元素
boolean addAll(Colletion<? extends E> c)  //把Collection c中的所有元素添加到指定的Set里
void clear()                                  //清除Set中的元素，长度变为0
boolean contains(E e)                          //返回Set中是否包含指定元素
int hashCode()                                //返回此Set的哈希码值。
boolean isEmpty()                             //Set是否为空
Iterator<E> iterator()                         //返回一个Iterator对象，用于遍历Set中的元素
boolean remove(E e)                         //删除Set中指定的对象
default boolean removeIf(Predicate<? super E> filter) //删除此Set中满足给定谓词的所有元素
int size()                                    //返回Set里元素的个数
Object[] toArray()                            //把Set转化为一个数组
<T> T[] toArray(T[] a)                        //把Set转化为一个指定类型的数组，推荐使用此种方式
```

5. SortedSet接口主要方法

```
Comparator<? super E> comparator( )//返回调用SortedSet的比较比较器。如果SortedSet的元素按照自然顺序排序，则返回null。
E first( )                          //返回该SortedSet的第一个元素。
E last( )                           //返回调用SortedSet的最后一个元素。
```

6. NavigableSet接口

```
 E  ceiling(E e) //返回此 Set 中大于等于给定元素的最小元素；如果不存在这样的元素，则返回 null。

Iterator<E> descendingIterator() //返回在此 Set 的元素上的逆序迭代器。

E   floor(E e) //返回此 Set 中小于等于给定元素的最大元素；如果不存在这样的元素，则返回 null。

E   higher(E e) //返回此 Set 中严格大于给定元素的最小元素；如果不存在这样的元素，则返回 null。

E   lower(E e) //返回此 Set 中严格小于给定元素的最大元素；如果不存在这样的元素，则返回 null。

E   pollFirst() //获取并移除第一个（最低）元素；如果此 Set 为空，则返回 null。

E   pollLast() //获取并移除最后一个（最高）元素；如果此 Set 为空，则返回 null。
```