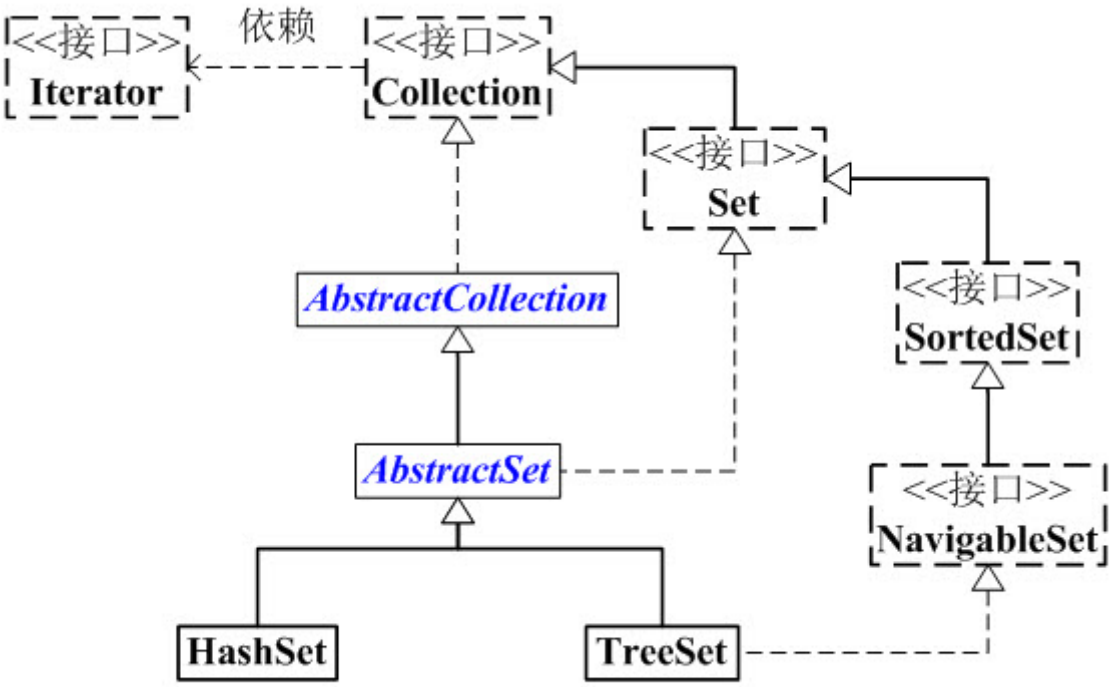
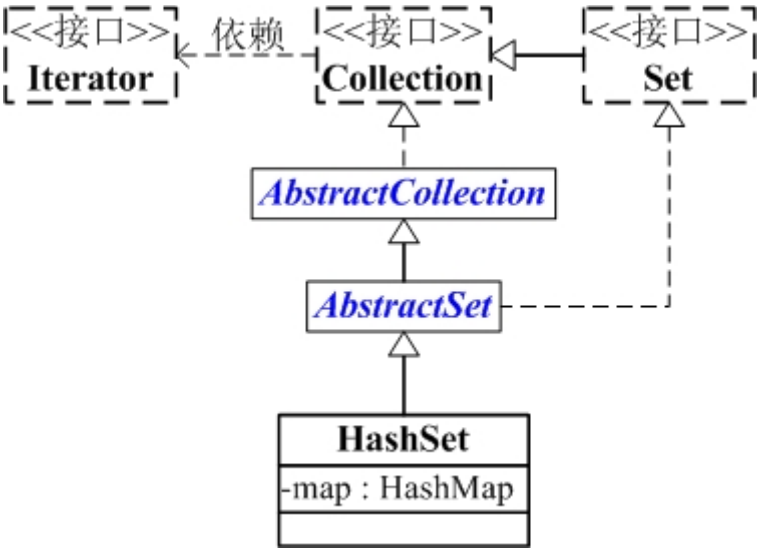
无序（即无索引）、不可重复。

Set的实现类都是基于Map来实现的(如，HashSet是通过HashMap实现的，TreeSet是通过TreeMap实现的（有序）)。Set的API和Collection完全一样。Set常用的实现类有：HashSet、TreeSet、LinkedHashSet。



HashSet

HashMap实现的，无重复，存储无序，可存储null元素。非同步。add就是在map中增加一个键值对，键对象就是这个元素，值对象是名为PRESENT的静态Object对象。哈希表是通过hashCode和equals方法来共同保证元素的唯一性。



HashSet中含有一个"HashMap类型的成员变量"map，HashSet的操作函数.

存储过程：

根据存储的元素计算出hashCode值，然后根据计算得出的hashCode值和数组的长度进行计算出存储的下标；如果下标的位置无元素，那么直接存储（hashcode()方法返回不相等，会存储在不同的位置，不管equal()方法的返回）；如果有元素，那么使用要存入的元素和该元素进行equals方法，如果结果为真，则已经有相同的元素了，所以直接不存。如果结果假，那么进行存储，以链表的形式存储。

注意：

向HashSet集合中存储自定义对象时，为了保证set集合的唯一性，那么必须重写hashCode和equals方法。

equals ( ) 和hashCede ( )

equals() 定义在JDK的Object.java中。通过判断两个对象的地址是否相等(即，是否是同一个对象)。

Object.java中定义了equals()方法，意味着所有的Java类都实现了equals()方法， 但是，使用默认的“equals()”方法，等价

于“==”方法。我们也可以在Object的子类中重写此方法，自定义“equals()”方法，

hashCode() 的作用**是获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。**

hashCode() 定义在JDK的Object.java中，仅仅当创建某个“类"的散列表时，该类的hashCode() 才有用。就是创建包含该类的HashMap，Hashtable，HashSet集合时，hashCode() 才有用。在散列表中，hashCode()作用是：**确定该类的每一个对象在散列表中的位置**

遍历

4.1 通过Iterator遍历HashSet

第一步：**根据iterator()获取HashSet的迭代器。**

第二步：**遍历迭代器获取各个元素。**

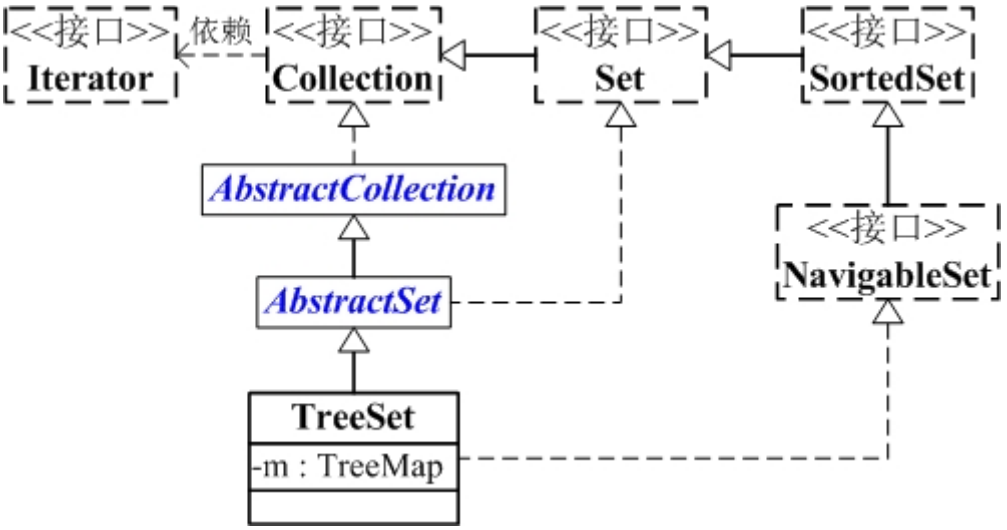
4.2 通过for-each遍历HashSet

第一步：**根据toArray()获取HashSet的元素集合对应的数组。**

第二步：**遍历数组，获取各个元素。**

TreeSet

TreeSet底层实际是用TreeMap实现的，通过key来存储Set的元素。 不能存储null，可对存储的元素进行排序。是SortedSet接口的实现类，



提供的额外方法：

comparator():返回对此 set 中的元素进行排序的比较器；如果此 set 使用其元素的自然顺序，则返回null。

first():返回此 set 中当前第一个（最低）元素。

last(): 返回此 set 中当前最后一个（最高）元素。

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。

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。

subSet(E fromElement, E toElement):返回此 set 的部分视图，其元素从 fromElement（包括）到 toElement（不包括）。

headSet(E toElement):返回此 set 的部分视图，其元素小于 toElement。

tailSet(E fromElement):返回此 set 的部分视图，其元素大于等于 fromElement。

Treeset的排序

自然排序

Java提供了一个Comparable接口，该接口里定义了一个compareTo(Object obj)方法可用来比较大小,以升序排列，该方法返回一个整数值。例：obj1.compareTo(obj2)

Java的一些常用类已经实现了Comparable接口，并提供了比较大小的标准。例如，String按字符串的UNICODE值进行比较，Integer等所有数值类型对应的包装类按它们的数值大小进行比较。

注意：TreeSet中只能添加同一种类型的对象，否则无法比较，会出现异常。对于TreeSet集合，判断两个对象是否相等的唯一标准是：两个对象通过compareTo(Object obj)方法比较是否返回0

定制排序

通过Comparator接口中int compare(T o1,T o2)方法，用于比较o1,o2的大小。实现定制排序，则需要在创建TreeSet时，调用一个带参构造器，传入Comparator对象。集合元素可以不必实现Comparable接口。例：

```
public static void main(String[] args){
        Person p1 = new Person();
        p1.age =20;
        Person p2 =new Person();
        p2.age = 30;
        Comparator<Person> comparator = new Comparator<Person>() {

            @Override
            public int compare(Person o1, Person o2) {
                //年龄越小的排在越后面
                if(o1.age<o2.age){
                    return 1;
                }else if(o1.age>o2.age){
                    return -1;
                }else{
                    return 0;
                }

            }
        };
        TreeSet<Person> set = new TreeSet<Person>(comparator);
        set.add(p1);
        set.add(p2);
        System.out.println(set);
```

```
    }
```

注意:如果向TreeSet中添加了一个可变对象后，并且后面程序修改了该可变对象的实例变量，这将导致它与其他对象的大小顺序发生了改变，但TreeSet不会再次调整它们。**推荐不要修改放入TreeSet集合中元素的关键实例变量。**

TreeSet和Comparable接口的使用

```java
public class Test {
    public static void main(String[] args) {
        User u1 = new User(1001, "高淇", 18);
        User u2 = new User(2001, "高希希", 5);
        Set<User> set = new TreeSet<User>();
        set.add(u1);
        set.add(u2);
    }
}

class User implements Comparable<User> {
    int id;
    String uname;
    int age;

    public User(int id, String uname, int age) {
        this.id = id;
        this.uname = uname;
        this.age = age;
    }
    /**
     * 返回0 表示 this == obj 返回正数表示 this > obj 返回负数表示 this < obj
     */
    @Override
    public int compareTo(User o) {
        if (this.id > o.id) {
            return 1;
        } else if (this.id < o.id) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

使用TreeSet要点：

(1) 非线程安全

(2) TreeSet中不能放入null元素。 TreeSet内部需要对存储的元素进行排序，自定义的类需要实现Comparable接口。否则抛出异常：java.lang.ClassCastException。重写compareTo()方法比较对象之间的大小进行内部排序。或在创建TreeSet的时候向构造器中传入比较器Comparator接口实现类对象，实现Comparator接口重写compara方法。

遍历

4.1 Iterator顺序遍历

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

4.2 Iterator顺序遍历

```
// 假设set是TreeSet对象
for(Iterator iter = set.descendingIterator(); iter.hasNext(); ) {
    iter.next();
}
```

4.3 for-each遍历HashSet

```
// 假设set是TreeSet对象，并且set中元素是String类型
String[] arr = (String[])set.toArray(new String[0]);
for (String str:arr)
    System.out.printf("for each : %s\n", str);
```

TreeSet不支持快速随机遍历，只能通过迭代器进行遍历！

LinkedHashSet

基于链表和哈希表(根据元素的hashcode值)共同实现的，存取有序，元素唯一。当遍历LinkedHashSet集合里的元素时，LinkedHashSet将会按元素的添加顺序来访问集合里的元素。元素重复的判断标准与HashSet一致。

EnumSet

专为枚举类设计的集合类，EnumSet中的所有元素都必须是指定枚举类型的枚举值，该枚举类型在创建EnumSet时显示或隐式地指定。EnumSet的集合元素也是有序的，EnumSet以枚举值在EnumSet类内的定义顺序来决定集合元素的顺序。非线程安全。

1.EnumSet集合不允许加入null元素。EnumSet中的所有元素都必须是指定枚举类型的枚举值。

2.EnumSet类没有暴露任何构造器来创建该类的实例，程序应该通过它提供的类方法来创建EnumSet对象。

方法：EnumSet没有其他额外增加的方法，只是增加了一些创建EnumSet对象的方法。

static <E extends Enum<E>> EnumSet<E>	**allOf**(Class<E> elementType) 创建一个包含指定元素类型的所有元素的枚举 set。
EnumSet<E>	**clone**() 返回 set 的副本。
static <E extends Enum<E>> EnumSet<E>	**complementOf**(EnumSet<E> s) 创建一个其元素类型与指定枚举 set 相同的枚举 set，最初包含指定 set 中所不包含的此类型的所有元素。
static <E extends Enum<E>> EnumSet<E>	**copyOf**(Collection<E> c) 创建一个从指定 collection 初始化的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**copyOf**(EnumSet<E> s) 创建一个其元素类型与指定枚举 set 相同的枚举 set，最初包含相同的元素（如果有的话）。
static <E extends Enum<E>> EnumSet<E>	**noneOf**(Class<E> elementType) 创建一个具有指定元素类型的空枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E e) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E first, E... rest) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E e1, E e2) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E e1, E e2, E e3) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E e1, E e2, E e3, E e4) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**of**(E e1, E e2, E e3, E e4, E e5) 创建一个最初包含指定元素的枚举 set。
static <E extends Enum<E>> EnumSet<E>	**range**(E from, E to) 创建一个最初包含由两个指定端点所定义范围内的所有元素的枚举 set。

总结

性能：**EnumSet性能>HashSet性能>LinkedHashSet>TreeSet性能**

EnumSet内部以位向量的形式存储，结构紧凑、高效，且只存储枚举类的枚举值，所以最高效。

HashSet以hash算法进行位置存储，特别适合用于添加、查询操作。

LinkedHashSet由于要维护链表，性能比HashSet差点，但是有了链表，

LinkedHashSet更适合于插入、删除以及遍历操作。

而TreeSet需要额外的红黑树算法来维护集合的次序，性能最次。

使用场景：

当需要一个特定排序的集合时，使用TreeSet集合。

当需要保存枚举类的枚举值时，使用EnumSet集合。

当经常使用添加、查询操作时，使用HashSet。

当经常插入排序或使用删除、插入及遍历操作时，使用LinkedHashSet。