一，hashset

1，hashset可以添加null值

2，①set存储的元素是无序的，是不可重复的，但是无序性并不等于随机性，真正的无序性，指的是

元素在底层存储的位置是无序的

②不可重复性，当向set中添加相同的元素的时候，后面的这个元素不能添加进去

说明：①要求添加进set集合的元素所在的类，重写equals（）方法和hashCode（）方法，进

而保证set中的元素的不可重复性

②set中的元素使用了哈希算法

③当向set中添加元素时，首先调用此对象所在类的hashCode（）方法，计算此对象的

哈希值，此哈希值决定了此对象在set中的存储位置，若此位置之前没有存储对象，

则每一个对象直接存储到此位置，如此位置已有对象存储，在通过equals（）方法比

较这俩个对象是否相同，如果相同，后一个对象就不在添加进来，

万一返回false：则要求hashCode()和equals()方法一致（不建议如此）

二，linkedhashset是hashset的子类

①linkedhashset使用了链表维护了一个添加元素在集合中的顺序，导致当我们遍历linkedhashset集合

元素时，是按添加进去的顺序进行遍历的 ，不能通过角标访问元素

②linkedhashset插入性能略低于hashset，但是迭代访问set里面的全部元素时有很好的性能

③linkedhashset不允许集合元素重复

三，treeset

1，向treeset中添加元素必须是同一个类的；

2，可以按照添加进集合中的元素的指定的顺序遍历，像string包装类等默认按照从小到大的顺序进行

遍历，按照字典顺序进行输出；

3，当添加的对象是一个自定义的类对象时，如果该类没有实现comparable接口，会报错

ClassCastException（当向treeset中添加自定义的类的对象时，有俩种排序方法，①自然排序②

定制排序）

4，自然排序，要求自定义类实现java.lang.Comparable接口，并重写comparable（object obj），在

此方法中指明按照自定义类的那个属性进行比较

5，向treeset中添加元素时，首先按照compareTo（）进行比较，一旦返回0，虽然仅仅是俩个对象的

此属性值相同，但是程序会认为这俩个对象是相同的，进而后一个对象就不能添加进来

compareTo（）和hashCode（）以及equals（）三者一致

自然排序：

```
System.out.println("treeset:");
        Set set2 = new  TreeSet();

        animal a3 = new animal("dfj",21);
        animal a4 = new animal("aaa",10);
        animal a5 = new animal("ccc",31);
        set2.add(a3);
        set2.add(a4);
        set2.add(a5);
        for (Object so:set2) {
                System.out.println(so);
        }


        @Override
        public int compareTo(Object o) {
                // TODO Auto-generated method stub
                if (o instanceof animal) {
                        animal  a1 = (animal)o;
                        return  (this.nameString.compareTo(a1.nameString));
                }
```

```
                return 0;
        }
```

treeset定制排序：

compare（）与hashCode（）与equals一致

```
public void testTreeset1(){
        //1，创建一个实现了comparator接口的对象
        Comparator comparator = new Comparator() {
                //2，向treeset中添加person对象，在此compare（）方法中指明按照哪个属性进行排序
                @Override
                public int compare(Object o1, Object o2) {
                        if (o1 instanceof person && o2 instanceof person) {
                                person p1 = (person)o1;
                                person p2 = (person)o2;
                                 int i = p1.getId().compareTo(p2.getId());
                                 if (i==0) {
                                return
p1.getNameString().compareTo(p2.getNameString());
                                }
                                 else {
                                        return i;
                                }
                        }
                        // TODO Auto-generated method stub
                        return 0;
                }
        };
        Set set = new TreeSet(comparator);

        set.add(new person("aa",123));
        set.add(new person("ee", 423));
        set.add( new person("cc", 124));
        set.add(new person("djf",121));
        for (Object object:set) {
```

```
                System.out.println(object);
        }
}
}
```

对于compareTo和equals两个方法我们可以总结为：compareTo是判断元素在排序中的位置是否相等，equals是判断元素是否相等，既然一个决定排序位置，一个决定相等，所以我们非常有必要确保当排序位置相同时，其equals也应该相等。