排序器[Ordering]是Guava流畅风格比较器[Comparator]的实现，它可以用来为构建复杂的比较器，以完成集合排序的功能

创建排序器：常见的排序器可以由下面的静态方法创建

方法	描述
natural()	对可排序类型做自然排序，如数字按大小，日期按先后排序
usingToString()	按对象的字符串形式做字典排序[lexicographical ordering]
from(Comparator)	把给定的Comparator转化为排序器

```
1. import com.google.common.collect.Lists;
2. import com.google.common.collect.Ordering;
3.
4. import java.util.List;
5.
6. public class OrderingTest1 {
7.     public static void main(String[] args) {
8.         List<String> stringList = Lists.newArrayList();
9.         stringList.add("luluyang");
10.         stringList.add("songyangjia");
11.         stringList.add("chagnhaha");
12.         stringList.add("buzhidaohaofanaaaaaaaa");
13.         stringList.add("wozhedebuzhidaoa");
14.
15.         //String str = "lili";
16.
17.         System.out.println("====================原List====================");
18.         System.out.println("striongList:" + stringList);
19.         System.out.println();
20.
21.         Ordering<String> natural = Ordering.natural();
22.         Ordering<Object> usingToString = Ordering.usingToString();
23.         Ordering<Object> arbitrary = Ordering.arbitrary();
24.
25.         System.out.println("====================natural（自然序）====================");
26.         stringList = natural.sortedCopy(stringList);
27.         System.out.println("natural:" + stringList);
28.         System.out.println();
29.         System.out.println("====================natural（字典序）====================");
30.         stringList = usingToString.sortedCopy(stringList);
31.         System.out.println("usingToString:" + stringList);
32.         System.out.println();
33.         System.out.println("====================natural（无序）====================");
34.         stringList = arbitrary.sortedCopy(stringList);
35.         System.out.println("arbirary:" + stringList);
36.         System.out.println();
37.
38.         Ordering<Integer> orderingBig = new Ordering<Integer>() {
39.             @Override
40.             public int compare(Integer left, Integer right) {
41.                 return left - right;
42.             }
43.         };
44.         List<Integer> integerList = Lists.newArrayList();
45.         integerList.add(3);
46.         integerList.add(4);
47.         integerList.add(5);
48.         integerList.add(7);
49.         integerList.add(2);
50.         integerList.add(9);
51.         integerList.add(8);
52.         integerList.add(1);
53.         integerList.add(0);
54.         integerList =  orderingBig.sortedCopy(integerList);
55.         System.out.println("interList:" + integerList);
56.     }
57. }
```

链式调用方法：通过链式调用，可以由给定的排序器衍生出其它排序器

方法	描述
reverse()	获取语义相反的排序器
nullsFirst()	使用当前排序器，但额外把null值排到最前面。
nullsLast()	使用当前排序器，但额外把null值排到最后面。
compound(Comparator)	合成另一个比较器，以处理当前排序器中的相等情况。
lexicographical()	基于处理类型T的排序器，返回该类型的可迭代对象Iterable<T>
onResultOf(Function)	对集合中元素调用Function，再按返回值用当前排序器排序。

例如，你需要下面这个类的排序器。

```
1. class Foo {
2.     @Nullable String sortedBy;
3.     int notSortedBy;
4. }
```

考虑到排序器应该能处理sortedBy为null的情况，我们可以使用下面的链式调用来合成排序器：

```
1. Ordering<Foo> ordering = Ordering.natural().nullsFirst().onResultOf(new Function<Foo, String>() {
2.   public String apply(Foo foo) {
3.     return foo.sortedBy;
4.   }
5. });
```

运用排序器：Guava的排序器实现有若干操纵集合或元素值的方法

方法	描述	另请参见
greatestOf(Iterable iterable, int k)	获取可迭代对象中最大的k个元素。	leastOf
isOrdered(Iterable)	判断可迭代对象是否已按排序器排序：允许有排序值相等的元素。	isStrictlyO
sortedCopy(Iterable)	判断可迭代对象是否已严格按排序器排序：不允许排序值相等的元素。	immutableSo
min(E, E)	返回两个参数中最小的那个。如果相等，则返回第一个参数。	max(E, E)
min(E, E, E, E...)	返回多个参数中最小的那个。如果有超过一个参数都最小，则返回第一个最小的参数。	max(E, E, E
min(Iterable)	返回迭代器中最小的元素。如果可迭代对象中没有元素，则抛出NoSuchElementException。	max(Iterabl