**[Java中的排序比较方式：自然排序和比较器排序](http://crazylittle-163-com.iteye.com/blog/513900)**

[Java](http://www.iteye.com/blogs/tag/Java)[算法](http://www.iteye.com/blogs/tag/%E7%AE%97%E6%B3%95)[JDK](http://www.iteye.com/blogs/tag/JDK)[J#](http://www.iteye.com/blogs/tag/J%23)

　　这里所说到的Java中的排序并不是指插入排序、希尔排序、归并排序等具体的排序算法。而是指执行这些排序算法时，比较两个对象“大小”的比较操作。我们很容易理解整型的 i>j 这样的比较方式，但当我们对多个对象进行排序时，如何比较两个对象的“大小”呢？这样的比较 stu1 > stu2 显然是不可能通过编译的。为了解决如何比较两个对象大小的问题，JDK提供了两个接口 java.lang.Comparable 和 java.util.Comparator 。   
  
一、自然排序：java.lang.Comparable   
　　Comparable 接口中只提供了一个方法： compareTo(Object obj) ，该方法的返回值是 int 。如果返回值为正数，则表示当前对象(调用该方法的对象)比 obj 对象“大”；反之“小”；如果为零的话，则表示两对象相等。下面是一个实现了 Comparable 接口的 Student 类：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Student **implements** Comparable {
3. **private** **int** id;
5. **private** String name;
7. **public** Student() {
8. **super**();
9. }
11. @Override
12. **public** **int** compareTo(Object obj) {
13. **if** (obj **instanceof** Student) {
14. Student stu = (Student) obj;
15. **return** id - stu.id;
16. }
17. **return** 0;
18. }
20. @Override
21. **public** String toString() {
22. **return** "<" + id + ", " + name + ">";
23. }
24. }

　　Student 实现了自然排序接口 Comparable ，那么我们是怎么利用这个接口对一组 Student 对象进行排序的呢？我们在学习数组的时候，使用了一个类来给整型数组排序： java.util.Arrays 。我们使用 Arrays 的 sort 方法来给整型数组排序。翻翻 API 文档就会发现， Arrays 里给出了 sort 方法很多重载形式，其中就包括 sort(Object[] obj) ，也就是说 Arryas 也能对对象数组进行排序，排序过程中比较两个对象“大小”时使用的就是 Comparable 接口的 compareTo 方法。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** CompareTest {
3. **public** **static** **void** main(String[] args) {
4. Student stu1 = **new** Student(1, "Little");
5. Student stu2 = **new** Student(2, "Cyntin");
6. Student stu3 = **new** Student(3, "Tony");
7. Student stu4 = **new** Student(4, "Gemini");
9. Student[] stus = **new** Student[4];
10. stus[0] = stu1;
11. stus[1] = stu4;
12. stus[2] = stu3;
13. stus[3] = stu2;
14. System.out.println(“Array: ” + Arrays.toString(stus));
15. Arrays.sort(stus);
16. System.out.println(“Sort:  ” + Arrays.toString(stus));
17. }
18. }

　　Student 数组里添加元素的顺序并不是按学号 id 来添加的。调用了 Arrays.sort(stus) 之后，对 Student 数组进行排序，不管 sort 是使用哪种排序算法来实现的，比较两个对象“大小”这个操作，它是肯定要做的。那么如何比较两个对象的“大小”？ Student 实现的 Comparable 接口就发挥作用了。 sort 方法会将待比较的那个对象强制类型转换成 Comparable ，并调用 compareTo 方法，根据其返回值来判断这两个对象的“大小”。所以，在这个例子中排序后的原 Student 乱序数组就变成了按学号排序的 Student 数组。   
  
　　但是我们注意到，排序算法和 Student 类绑定了， Student 只有一种排序算法。但现实社会不是这样的，如果我们不想按学号排序怎么办？假如，我们想按姓名来给学生排序怎么办？我们只能修改 Student 类的 Comparable 接口的 compareTo 方法，改成按姓名排序。如果在同一个系统里有两个操作，一个是按学号排序，另外一个是按姓名排序，这怎么办？不可能在 Student 类体中写两个 compareTo 方法的实现。这么看来Comparable就有局限性了。为了弥补这个不足，JDK 还为我们提供了另外一个排序方式，也就是下面要说的比较器排序。   
  
二、比较器排序：java.util.Comparator   
　　上面我提到了，之所以提供比较器排序接口，是因为有时需要对同一对象进行多种不同方式的排序，这点自然排序 Comparable 不能实现。另外， Comparator 接口的一个好处是将比较排序算法和具体的实体类分离了。   
  
　　翻翻 API 会发现， Arrays.sort 还有种重载形式：sort(T[] a, Comparator<? super T> c) ，这个方法参数的写法用到了泛型，我们还没讲到。我们可以把它理解成这样的形式： sort(Object[] a, Comparator c) ，这个方法的意思是按照比较器 c 给出的比较排序算法，对 Object 数组进行排序。Comparator 接口中定义了两个方法： compare(Object o1, Object o2) 和 equals 方法，由于 equals 方法所有对象都有的方法，因此当我们实现 Comparator 接口时，我们只需重写 compare 方法，而不需重写 equals 方法。Comparator 接口中对重写 equals 方法的描述是：“注意，不重写 Object.equals(Object) 方法总是安全的。然而，在某些情况下，重写此方法可以允许程序确定两个不同的 Comparator 是否强行实施了相同的排序，从而提高性能。”。我们只需知道第一句话就OK了，也就是说，可以不用去想应该怎么实现 equals 方法，因为即使我们不显示实现 equals 方法，而是使用Object类的 equals 方法，代码依然是安全的。而对于第二句话，究竟是怎么提高比较器性能的，我也不了解，所以就不说了。   
  
　　那么我们来写个代码，来用一用比较器排序。还是用 Student 类来做，只是没有实现 Comparable 接口。由于比较器的实现类只用显示实现一个方法，因此，我们可以不用专门写一个类来实现它，当我们需要用到比较器时，可以写个匿名内部类来实现 Comparator 。下面是我们的按姓名排序的方法：

**Java代码  [收藏代码](javascript:void())**

1. **public** **void** sortByName () {
2. Student stu1 = **new** Student(1, "Little");
3. Student stu2 = **new** Student(2, "Cyntin");
4. Student stu3 = **new** Student(3, "Tony");
5. Student stu4 = **new** Student(4, "Gemini");
7. Student[] stus = **new** Student[4];
8. stus[0] = stu1;
9. stus[1] = stu4;
10. stus[2] = stu3;
11. stus[3] = stu2;
12. System.out.println("Array: " + Arrays.toString(stus));
14. Arrays.sort(stus, **new** Comparator() {
16. @Override
17. **public** **int** compare(Object o1, Object o2) {
18. **if** (o1 **instanceof** Student && o2 **instanceof** Student) {
19. Student s1 = (Student) o1;
20. Student s2 = (Student) o2;
21. //return s1.getId() - s2.getId(); // 按Id排
22. **return** s1.getName().compareTo(s2.getName()); // 按姓名排
23. }
24. **return** 0;
25. }
27. });
29. System.out.println("Sorted: " + Arrays.toString(stus));
30. }

　　当我们需要对Student按学号排序时，只需修改我们的排序方法中实现Comparator的内部类中的代码，而不用修改 Student 类。   
  
　　P.S. 当然，你也可以用 Student 类实现 Comparator 接口，这样Student就是（is a）比较器了（Comparator）。当需要使用这种排序的时候，将 Student 看作 Comparator 来使用就可以了，可以将 Student 作为参数传入 sort 方法，因为 Student is a Comparator 。但这样的代码不是个优秀的代码，因为我们之所以使用比较器（Comparator），其中有个重要的原因就是，这样可以把比较算法和具体类分离，降低类之间的耦合。   
  
　　上一篇博客里说到了，TreeSet对这两种比较方式都提供了支持，分别对应着TreeSet的两个构造方法：   
　　　　1、TreeSet()：根据TreeSet中元素实现的 Comparable 接口的 compareTo 方法比较排序   
　　　　2、TreeSet(Comparator comparator)：根据给定的 comparator 比较器，对 TreeSet 中的元素比较排序  
　　当向 TreeSet 中添加元素时，TreeSet 就会对元素进行排序。至于是用自然排序还是用比较器排序，就看你的 TreeSet 构造是怎么写的了。当然，添加第一个元素时不会进行任何比较， TreeSet 中都没有元素，和谁比去啊？   
  
　　下面，分别给出使用两种排序比较方式的 TreeSet 测试代码：

**Java代码  [收藏代码](javascript:void())**

1. /\*\*
2. \* 使用自然排序
3. \* Student必须实现Comparable接口，否则会抛出ClassCastException
4. \*/
5. **public** **void** testSortedSet3() {
6. Student stu1 = **new** Student(1, "Little");
7. Student stu2 = **new** Student(2, "Cyntin");
8. Student stu3 = **new** Student(3, "Tony");
9. Student stu4 = **new** Student(4, "Gemini");
11. SortedSet set = **new** TreeSet();
12. set.add(stu1);
13. set.add(stu3); // 若Student没有实现Comparable接口，抛出ClassCastException
14. set.add(stu4);
15. set.add(stu2);
16. set.add(stu4);
17. set.add(**new** Student(12, "Little"));
19. System.out.println(set);
20. }

**Java代码  [收藏代码](javascript:void())**

1. /\*\*
2. \* 使用比较器排序
3. \* Student可以只是个简单的Java类，不用实现Comparable接口
4. \*/
5. **public** **void** testSortedSet3() {
6. Student stu1 = **new** Student(1, "Little");
7. Student stu2 = **new** Student(2, "Cyntin");
8. Student stu3 = **new** Student(3, "Tony");
9. Student stu4 = **new** Student(4, "Gemini");
11. SortedSet set = **new** TreeSet(**new** Comparator() {
13. @Override
14. **public** **int** compare(Object o1, Object o2) {
15. **if** (o1 **instanceof** Student
16. && o2 **instanceof** Student) {
17. Student s1 = (Student) o1;
18. Student s2 = (Student) o2;
19. **return** s1.getName().compareTo(s2.getName());
20. }
21. **return** 0;
22. }
24. });
26. set.add(stu1);
27. set.add(stu3);
28. set.add(stu4);
29. set.add(stu2);
30. set.add(stu4);
31. set.add(**new** Student(12, "Little"));
33. System.out.println(set);
34. }

　　另外，介绍个工具类，java.util.Collections。注意，这不是Collection接口。Collections很像Arrays类。Arrays提供了一系列用于对数组操作的静态方法，查找排序等等。Collections也提供了一系列这样的方法，只是它是用于处理集合的，虽然Collections类和Collection接口很像，但是不要被Collections的名字给欺骗了，它不是只能处理Collection接口以及子接口的实现类，同样也可以处理Map接口的实现类。