1. 比较器Comparable
   1. Comparable接口中只有一个方法：

public int compareTo(T o);

调用此方法的对象，也就是this和o进行比较，若返回值大于0则this大于o，返回值等于0则是this等于o，返回值小于0则是this<o,而这个Comparable是直接在我们的自定义类User上实现，因为this是需要一个明确的比较对象的，也就是一般情况下我们会在定义User类的时候有排序的需求，就要实现此接口。

范例：

|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class UserComparable implements Comparable<UserComparable> {  private String name;  private int age;  public UserComparable(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public String toString() {  return "UserComparable{" +  "name='" + name + '\'' +  ", age=" + age +  '}';  }  @Override  public int compareTo(UserComparable o) {  //由于字符串无法直接比较大小，所以调用String类的compareTo  if (this.name.compareTo(o.name)==0){  if (this.age == o.age){  return 0;  }else if (this.age >o.age){  return 1;  }else {  return -1;  }  }else if (this.name.compareTo(o.name)>0){  return 1;  }else {  return -1;  }  }  public static void main(String[] args) {  List<UserComparable> list = new ArrayList<UserComparable>();  list.add(new UserComparable("gol",21));  list.add(new UserComparable("gol",19));  list.add(new UserComparable("xiao",21));  list.add(new UserComparable("long",21));  System.out.println("排序前："+list);  //排序规则：先按name排序，若name相等则再比较age  Collections.sort(list);  System.out.println("排序后："+list);  }  } |

1. Comparator：

Comparator接口中方法很多，但是我们只需要实现一个，也是最重要的一个compare，也许有的人会好奇为什么接口中的方法可以不用实现，因为这是JDK8以后的新特性，在接口中用default修饰的方法可以有方法体，在实现接口的时候可以不用重写，可以类比抽象类。

·int compare(T o1, T o2);

compare比较的o1和o2，返回值大于0则o1大于o2，依次类推，对于compare来说this是谁不重要，所比较的两个对象都已经传入到方法中，所以Comparator就是个外部比较器，在我们设计User初时，并不需要它有比较功能，在后期扩展业务是，Comparator的存在可以使我们在不修改源代码的情况下来完成需求,只需要新定义一个比较器来实现Comparator，重写compare方法并将User对象传进去。

也就是说Comparetor类中的方法不需要再设计之初就存在类中，他是一个挽救比较器，只需要重新定义一个比较工具类来继承它并覆写方法，然后再调用该方法进行比较。它定义的是一种规则。

范例：

|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  import java.util.List;  public class User {  private String name;  private int age;  public User(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public String toString() {  return "User{" +  "name='" + name + '\'' +  ", age=" + age +  '}';  }  public static void main(String[] args) {  List<User> list = new ArrayList<User>();  list.add(new User("gol", 21));  list.add(new User("gol", 19));  list.add(new User("xiao", 21));  list.add(new User("long", 21));  System.out.println("排序前：" + list);  //排序规则：先按name排序，若name相等则再比较age  //创建比较器对象  Comparator comparator = new UserComparator();  Collections.sort(list,comparator);  //Arrays.sort(goods,comparator);  //Collections.sort(coll,comparator);  //new TreeSet(comparator);  System.out.println("排序后：" + list);  }  static class UserComparator implements Comparator<User> {  @Override  public int compare(User o1, User o2) {  if (o1.name.compareTo(o2.name) == 0) {  if (o1.age == o2.age) {  return 0;  } else if (o1.age > o2.age) {  return 1;  } else {  return -1;  }  } else if (o1.name.compareTo(o2.name) > 0) {  return 1;  } else {  return -1;  }  }  }  } |

结论：  
1、Comparator 使用比较灵活，不需要修改实体类源码，但是需要实现一个比较器。  
2、Comparable 使用简单，但是对代码有侵入性，需要修改实体类源码。  
3、java中大部分我们常用的数据类型的类都实现了Comparable接口，而仅仅只有一个抽象类RuleBasedCollator实现了Comparator接口 ，还是我们不常用的类，  
这并不是说要用Comparab而不要使用Comparator，在设计初时有需求就选择Comparable，若后期需要扩展或增加排序需求是，再增加一个比较器Comparator，毕竟能写Collections.sort(arg1),没人乐意写Collections.sort(arg1,arg2)。