**JAVA基础**

# 集合-List,Set

## Collection

* 集合与数组的区别:

*1.数组元素可以是基本类型值也可以是对象，集合里只能保存对象（即不能保存基本数据类型数据）*

*2.数组长度不可改变，而集合没有长度约束*

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collection;  **import** java.util.Iterator;  **public** **class** CollectionDemo {  **public** **static** **void** main(String[] args) {  // ctrl + shift + o导包  // ArrayList extends AbstractList implements List  // List extends Collection(接口继承)  // 黄色波浪线表示的是警告，这里没有指定泛型，所以有黄色波浪线警告  Collection collection = **new** ArrayList();  // 添加元素,添加成功返回true  System.***out***.println(collection.add("123"));  System.***out***.println(collection.add(**new** Student("张三","上海市")));  // 集合的大小，即集合添加了多少个元素  System.***out***.println(collection.size());  // 输出集合信息  System.***out***.println(collection.toString());  // 将一个集合添加到另一个调用的集合中  System.***out***.println(collection.addAll(collection));    System.***out***.println(collection.size());// 4    collection.clear();  System.***out***.println(collection.size());// 0    System.***out***.println(**null** == collection);// false    System.***out***.println(collection.isEmpty());// true    System.***out***.println(collection.add(**new** Student("张三","上海市")));  // false 集合中存储的是对应的引用，这里通过new创建出来的内容，虽然在值上相同  // 但是只要new就会在堆中重新分配地址空间，引用(地址)自然不同  System.***out***.println(collection.contains(**new** Student("张三","上海市")));    Student stu = **new** Student("李四","苏州市");  // 此时集合中共2个元素  collection.add(stu);    // true 前后两个对象地址相同  System.***out***.println(collection.contains(stu));  // 不可以删除，因为他们地址不一致  System.***out***.println(collection.remove(**new** Student("李四","苏州市")));  // 可以删除，因为添加与删除时同一个对象，地址相同  // System.out.println(collection.remove(stu));  System.***out***.println(collection.size());    System.***out***.println(collection.add("123"));    // 不要只看表面上的实现，一定要挖掘出他内部的实现逻辑，  // 只有这样在发生问题时，才能够很好分析与定位问题  // 即这里的for-each循环中的obj实际调用的是其指向子类的toString方法  Object[] objs = collection.toArray();  **for**(Object obj:objs) {  System.***out***.println(obj.toString());  }  System.***out***.println("=========================");  // 迭代器， Iterator遍历集合,以后会经常用到,只能向后遍历  Iterator iterator = collection.iterator();  // hasNext是否有下一个元素，有则返回true，否则返回false  // next指针移到下一个元素，并返回相应的元素值  **while**(iterator.hasNext()) {    Object obj = iterator.next();  **if** (obj **instanceof** Student) {  Student stu1 = (Student) obj;  stu1.setName("王五");  }  }    // 因为iterator只能向后遍历，所以在此需要新建一个迭代器对象，  // 而不能使用原来的迭代器对象，就是上面的iterator  Iterator iterator1 = collection.iterator();  **while**(iterator1.hasNext()) {  System.***out***.println("-----");  System.***out***.println(iterator1.next());  }      System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  // for-each循环遍历  **for**(Object obj:collection) {  System.***out***.println(obj);  }    }  } |

## ArrayList

*List:有序(添加顺序与存储顺序相同)、允许重复的集合*

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.ListIterator;  **public** **class** GenericColecDemo {  **public** **static** **void** main(String[] args) {  // alt + shift + r重命名变量，会修改同一作用域的所有这个变量  List<Student> list = **new** ArrayList<Student>();  // 对象的默认值就是null，所以可以添加  list.add(**null**);  System.***out***.println(list.size());    list.add(**new** Student("张三", "上海市"));  System.***out***.println(list.size());    list.clear();  System.***out***.println(list.isEmpty());  Student stu = **new** Student("李四", "上海市");  // 可以添加重复元素  list.add(stu);  list.add(stu);    list.add(**new** Student("张三", "上海市"));  list.add(**new** Student("王五", "苏州市"));    System.***out***.println(list.size());    // ArrayList是有序集合，什么是有序？  // 添加顺序与存储顺序一致（遍历顺序）  // Iterator<Student> ite = list.iterator();  // ListIterator extends Iterator  ListIterator<Student> ite = list.listIterator();  **while**(ite.hasNext()) {  System.***out***.println(ite.next());  }    System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  // hasPrevious前面是否有元素  **while**(ite.hasPrevious()) {  // 指向前一个，并获取元素  System.***out***.println(ite.previous());  }  System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");      System.***out***.println(list.get(0));  // for循环遍历  **for** (**int** i = 0; i < list.size(); i++) {  System.***out***.println(list.get(i));  }    // for-each循环遍历自己写  System.***out***.println("===================");  // 下标从0开始,删除指定位置的元素，返回被删除的元素  // 当下标>=数组大小时，报出java.lang.IndexOutOfBoundsException  // Student stu1 = list.remove(10);  Student stu1 = list.remove(2);  System.***out***.println(stu1);    System.***out***.println("====================");  Student stu2 = **new** Student("赵六", "南京市");  // 下标大于集合的大小或小于0会抛出异常IndexOutOfBoundsException  list.add(3, stu2);// insert  **for** (**int** i = 0; i < list.size(); i++) {  System.***out***.println(list.get(i));  }      System.***out***.println(list.indexOf(stu2));// 3  System.***out***.println(list.indexOf(**new** Student("赵六", "南京市")));// -1      Student stu3 = **new** Student("刘七", "南京市");  list.set(3, stu3);// replace    System.***out***.println("====================");    **for** (**int** i = 0; i < list.size(); i++) {  System.***out***.println(list.get(i));  }      // 集合中常用  // 方法：get size add remove clear isEmpty Iterator(ListIterator、for、for-each)  // 注意判断的先后顺序（&&的短落操作）  **if** (**null** != list && !list.isEmpty()) {  // 对集合操作  }    }  } |

## HashSet

*Set:无序(添加顺序与存储顺序不同，hashcode/地址)、不可重复的集合*

Set集合不允许包含相同的元素，如果试图把两个相同的元素加入同一个Set集合中，则添加失败，add方法返回false，且新元素不会被加入

HashSet按Hash算法来存储集合中的元素，因此具有很好的存取和查找性能

* HashSet集合判断两个元素相等的标准是：

两个对象的hashCode()方法返回值相等，并且两个对象通过equals方法比较相等。

**注意：**如果需要把一个对象放入HashSet中时，如果重写该对象对应类的equals()方法，也应该重写其hashCode()方法，其规则是:如果2个对象通过equals方法比较返回true时，这两个对象的hashCode也应当相同

|  |
| --- |
| **public** **class** Student **implements** Comparable<Student>{    **private** String name;  **private** String address;  **private** **int** sno;      **public** Student() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** Student(String name, String address) {  **super**();  **this**.name = name;  **this**.address = address;  }      **public** Student(String name, String address, **int** sno) {  **super**();  **this**.name = name;  **this**.address = address;  **this**.sno = sno;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getAddress() {  **return** address;  }  **public** **void** setAddress(String address) {  **this**.address = address;  }    // @Override  // public int hashCode() {  // final int prime = 31;  // int result = 1;  // result = prime \* result + ((address == null) ? 0 : address.hashCode());  // result = prime \* result + ((name == null) ? 0 : name.hashCode());  // return result;  // }  // @Override  // public boolean equals(Object obj) {  // if (this == obj)  // return true;  // if (obj == null)  // return false;  // if (getClass() != obj.getClass())  // return false;  // Student other = (Student) obj;  // if (address == null) {  // if (other.address != null)  // return false;  // } else if (!address.equals(other.address))  // return false;  // if (name == null) {  // if (other.name != null)  // return false;  // } else if (!name.equals(other.name))  // return false;  // return true;  // }    // 添加元素时自动执行，不用手动调用  // 直接返回一个整数值，升序为正降序为负，升序降序是以添加元素顺序为基准  //  @Override  **public** **int** compareTo(Student o) {  // System.out.println("---" + this);  // 根据实体属性进行排序，比较时this在前表示升序，this在后表示降序  **return** **this**.sno-o.sno;// o.name.compareTo(this.name)  }  @Override  **public** String toString() {  **return** "Student [name=" + name + ", address=" + address + ", sno=" + sno + "]";  }      // 10个人的团队，学生类已经有了排序规则，为大家都来调用  // 现在你有个需求，不要默认已经提供的排序，需要一个按照名字的排序  // 现在怎么做？  // 改默认排序 （一定不行，这么做没有团队意识）  // 创建treeset自定义排序（可取的）    } |
| /\*\*  \* Set集合不允许包含相同的元素，如果试图把两个相同的元素加入同一个 Set集合中，则添加失败，add方法返回false，且新元素不会被加入  \*  \* HashSet按Hash算法来存储集合中的元素(为什么是无序的)，因此具有很好的存取和查找性能  \*  \* HashSet集合元素可以是null  \*  \* **@author** tzhang  \*  \*/  **public** **class** HashSetDemo {  **public** **static** **void** main(String[] args) {  // 每次new都会新开辟存储空间存储对象  Student stu1 = **new** Student("张三", "上海市");  Student stu2 = **new** Student("张三", "上海市");  // HashSet extends AbstractSet implements Set  // Set extends Collection  Set<Student> setStu = **new** HashSet<Student>();// LinkedHashSet  // 没有元素，可以直接添加成功  setStu.add(stu1);  // 会将stu2元素的hashcode值与已存在的元素的hashcode值进行比较  // 如果hashcode值与他们都不相同，就直接添加进来，添加成供  // 如果hashcode值与他们比较，有与他相同的值，那么再比较这两个元素的equals方法  // 如果equals方法返回值为true,那么就添加失败  // 如果equals方法返回值为false,那么就添加成功，  // （hashcode值相同,但是equals返回false，这种情况尽量不要发生）  // 这样会导致HashSet性能下降  setStu.add(stu2);  // 赋值操作不会在堆中新开辟存储空，只是将变量的引用指向了一块已经存在的空间  Student stu3 = stu2;  setStu.add(stu3);  System.***out***.println(setStu.size());  // 为什么没有与索引相关的操作方法？  // 因为set是存储是无序的，无序就是添加的顺序与遍历输出的顺序不一致  Student stu4 = **new** Student("王五", "上海市");  Student stu5 = **new** Student("赵六", "上海市");  Student stu6 = **new** Student("刘七", "上海市");  Student stu7 = **new** Student("李四", "上海市");  setStu.add(stu4);  setStu.add(stu5);  setStu.add(stu6);  setStu.add(stu7);  System.***out***.println("==========================");  **for**(Student stu:setStu){  // 观察输出数据，可以看到添加顺序与输出顺序不一致  System.***out***.println(stu);  }  System.***out***.println("==========================");  Iterator<Student> ite = setStu.iterator();  **while**(ite.hasNext()) {  System.***out***.println(ite.next());  }  }  } |

## TreeSet

ClassCastException

* TreeSet判断两个对象不相等的标准是:

*1.两个对象通过equals方法比较返回false*

*2.通过compareTo(Object obj)比较没有返回0(即使是同一个对象，compareTo方法返回0就认为他们相等)*

向TreeSet中添加的应该是同一个类的对象，否则也传给引发上面的异常，自己在写的时候要指定泛型，里面如果添加不同类型的对象会引发ClassCastException

**注意:**如果试图把一个对象添加进TreeSet时，则该对象的类必须实现Comparable接口，否则程序将会抛出ClassCastException异常。

* 定制排序

TreeSet的自然排序是根据集合元素的大小，TreeSet将它们以升序排列。如果需要实现定制排序，如降序，则可以使用Comparator接口的帮助。

Comparator接口中包含一个int compareTo(T o1,T o2)方法以，用于比较o1和o2的大小：

*1.返回正整数，则表明o1大于o2；*

*2.返回零，则表明o1等于o2*

*3.返回负整数，则表明o1小于o2*

如果需要实现定制排序，则需要在创建TreeSet集合对象时，并提供一个Comparator对象与该TreeSet集合关联，由该Comparator对象负责集合元素的排序逻辑。

|  |
| --- |
| **import** java.util.Comparator;  **import** java.util.Iterator;  **import** java.util.SortedSet;  **import** java.util.TreeSet;  **public** **class** TreeSetDemo {  **public** **static** **void** main(String[] args) {  // TreeSet extends AbstractSet implements NavigableSet  // NavigableSet extends SortedSet extends Set  // 可以自定义排序规则的集合，排序的对象要实现Compareable接口  SortedSet<Student> setStu = **new** TreeSet<Student>(**new** Comparator<Student>() {  // 定制排序  @Override  **public** **int** compare(Student o1, Student o2) {    **return** o1.getName().compareTo(o2.getName());  }  });    Student stu1 = **new** Student("a", "上海市", 11);  Student stu2 = **new** Student("d", "上海市",30);  Student stu3 = **new** Student("c", "上海市",8);  Student stu4 = **new** Student("f", "上海市",5);  Student stu5 = **new** Student("b", "上海市",50);    setStu.add(stu1);  //添加stu2时，执行stu2.compareTo(stu1)，因为默认返回的是1，所以stu2>stu1  // 因为按照自然排序，即升序排列，stu1 stu2  setStu.add(stu2);    // 添加stu3时，执行stu3.compareTo(stu1),因为默认返回的是1，所以stu3>stu1  // 再执行stu3.compareTo(stu2)，,因为默认返回的是1，所以stu3>stu2  // 所以treeset排列顺序是stu1 stu2 stu3,依次类推（每个比较2次，里面已做二分查找优化）  setStu.add(stu3);  setStu.add(stu4);  setStu.add(stu5);    Iterator<Student> iterator = setStu.iterator();  **while**(iterator.hasNext()) {  System.***out***.println(iterator.next());  }    }  } |

作业：  
1、使用集合改写学生管理系统  
2、实现简单的购物车功能（初始化20个商品，实现购物车的增删查改，即总价的显示）  
商品属性：商品ID，商品名，商品价格