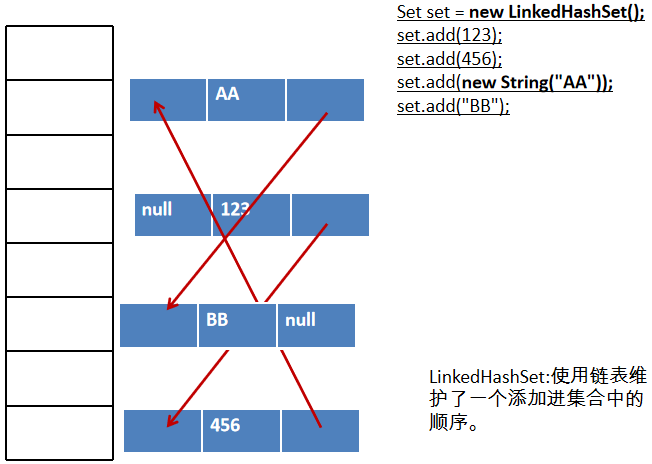
* **Set接口**
* Set接口是Collection的子接口，set接口没有提供额外的方法。
* **Set集合不允许包含相同的元素**，如果试把两个相同的元素加入同一个Set集合中，则添加操作失败。
* Set判断两个对象是否相同不是使用==运算符，而是**根据equals方法**。
* Set中的元素是如何存储的呢？使用了哈希算法。
* 当向Set中添加对象时，首先调用此对象所在类的hashCode()方法，计算此对象的哈希值，此哈希值决定了此对象在Set中的存储位置。
* 若此位置之前没有对象存储，则这个对象直接存到这个位置；若此位置已有对象存储，再通过equals()方法比较这两个对象是否相同。
* 如果相同，后一个对象就不能再添加进来；
* **Set的实现类之一：HashSet——常用**
* HashSet是Set接口的典型实现。
* 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具有以下特点：**
* 不能保证元素的排列顺序，无序性不等于随机性。
* HashSet不是线程安全的
* 集合元素可以是null
* 当向HashSet集合中存入一个元素时，HashSet会调用该对象的hashCode()方法来得到该对象的hashCode值，然后根据hashCode值决定该对象在HashSet中的存储位置。
* **HashSet集合判断两个元素相等的标准**：两个对象通过**hashCode()**方法比较相等，并且两个对象的**equals()**方法返回值也相等。
* **hashCode()方法**
* 如果两个元素的equals()方法返回true，但它们的hashCode()返回值不相等，hashSet将会把它们存储在不同的位置，但依然可以添加成功。
* **对于存放在Set容器中的对象，对应的类一定要重写equals()和hashCode(Object obj)方法，以实现对象相等规则。**
* 重写hashCode()方法的基本原则：
* 在程序运行时，同一个对象多次调用hashCode()方法应该返回相同的值。
* 当两个对象的equals()方法比较返回true时，这两个对象的hashCode()方法的返回值也应相等。
* 对象中用作equals()方法比较的Field，都应该用来计算hashCode值。
* **Set实现类之二：LinkedHashSet**
* **LinkedHashSet是HashSet的子类**。
* LinkedHashSet根据元素的hashCode值来决定元素的存储位置，但它同时**使用链表维护元素的次序，这使得元素看起来是以插入顺序保存的**。
* LinkedHashSet**插入性能略低于**HashSet，但在迭代访问Set里的全部元素时有很好的性能。
* LinkedHashSet不允许集合元素重复。



* Set实现类之三：TreeSet
* 向TreeSet中**添加的元素必须是同一个类**。
* 当Person类没有实现Comparable接口时，会报错ClassCastException。
* 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遍历。
* TreeSet是SortedSet接口的实现类，TreeSet可以确保集合元素处于排序状态。
* Comparator comparator()
* Object first()
* Object last()
* Object lower(Object e)
* Object higher(Object e)
* SortedSet subSet(fromElement, toElement)
* SortedSet headSet(toElement)
* SortedSet tailSet(fromElement)
* TreeSet两种排序方法：自然排序和定制排序。默认情况下，TreeSet采用自然排序。
* **自然排序**

TreeSet会调用集合元素的compareTo(Object obj)方法来比较元素之间的大小关系，然后将集合元素按升序排列。

/\*\*  
 \* 当向TreeSet中添加Person类的对象时，依据此方法，确定按照哪个属性排列。  
 \*/  
@Override  
public int compareTo(Object o) {  
 if (o instanceof Person) {  
 Person p = (Person)o;  
 return this.name.compareTo(p.name);  
 }  
 // 返回0表示比较相同，不添加  
 return 0;  
}

* **如果试图把一个对象添加到TreeSet时，则该对象的类必须实现Comparable接口**。
* 实现Comparable的类必须实现compareTo(Object obj)方法，**两个对象即通过compareTo(Object obj)方法的返回值来比较大小**。
* Comparable的典型实现：
* BigDecimal、BigInteger以及所有的数值型对应的包装类：按它们对应的数值大小进行比较
* Character：按字符的unicode值来进行比较
* Boolean：true对应的包装类实例大于false对应的包装类实例
* String：按字符串中字符的unicode值进行比较
* Date、Time：后边的时间、日期比前面的时间、日期大
* 向TreeSet中添加元素时，只有第一个元素无须比较compareTo()方法，后面添加的所有元素都会调用compareTo()方法进行比较。
* **因为只有相同类的两个实例才会比较大小，所以向TreeSet中添加的应该是同一个类的对象**。
* 对于TreeSet集合而言，它判断两个对象是否相等的唯一标准是：两个对象通过compareTo(Object obj)方法**比较返回值**。
* 当需要把一个对象放入TreeSet中，重写该对象对应的equals()方法时，应**保证该方法与compareTo(Object obj)方法有一致的结果**：如果两个对象通过equals()方法比较返回true，则通过compareTo(Object obj)方法比较应返回0。

@Override  
public int compareTo(Object o) {  
 if (o instanceof Person) {  
 Person p = (Person)o;  
 //return this.name.compareTo(p.name);  
 //return this.age.compareTo(p.age);  
   
 int i = this.age.compareTo(p.age);  
 if (i == 0) {  
 return this.name.compareTo(p.name);  
 }else {  
 return i;  
 }  
 }  
 // 返回0表示比较相同，不添加  
 return 0;  
}

* **定制排序**
* TreeSet的自然排序是根据集合元素的大小，进行元素升序排列。如果需要定制排序，比如降序排列，可通过Comparator接口的帮助。需要重写compare(T o1,T o2)方法。
* 利用int compare(T o1,T o2)方法，比较o1和o2的大小：如果方法返回正整数，则表示o1大于o2；如果返回0，表示相等；返回负整数，表示o1小于o2。
* **要实现定制排序，需要将实现Comparator接口的实例作为形参传递给TreeSet的构造器**。
* 此时，仍然只能向TreeSet中**添加类型相同的对象**。否则发生ClassCastException异常。
* 使用定制排序判断两个元素相等的标准是：通过Comparator比较两个元素返回了0。

@Test  
public void test4() {  
 //1)创建一个实现了Comparator接口的类对象  
 Comparator comparator = new Comparator() {  
 //向TreeSet中添加Customer类的对象，在此compare()方法中，指明是按照Customer哪个属性排序的。  
 @Override  
 public int compare(Object o1, Object o2) {  
 if (o1 instanceof Customer && o2 instanceof Customer) {  
 Customer c1 = (Customer) o1;  
 Customer c2 = (Customer) o2;  
 int i = c1.getId().compareTo(c2.getId());  
 if (i == 0) {  
 return c1.getName().compareTo(c2.getName());  
 }  
 return i;  
 }  
 return 0;  
 }  
 };  
  
 //2）将此对象作为形参传递给TreeSet的构造器中  
 TreeSet set = new TreeSet(comparator);  
 //3）向TreeSet中添加Comparator接口中的compare方法中涉及的类的对象。  
 set.add(new Customer("AA", 1003));  
 set.add(new Customer("BB", 1002));  
 set.add(new Customer("GG", 1004));  
 set.add(new Customer("CC", 1001));  
 set.add(new Customer("DD", 1001));  
  
 //4）遍历  
 for (Object str : set) {  
 System.***out***.println(str);  
 }  
}

同下

@Test  
public void test5() {  
  
 TreeSet set = new TreeSet(new Comparator() {  
 @Override  
 public int compare(Object o1, Object o2) {  
 if (o1 instanceof Customer && o2 instanceof Customer) {  
 Customer c1 = (Customer) o1;  
 Customer c2 = (Customer) o2;  
 int i = c1.getId().compareTo(c2.getId());  
 if (i == 0) {  
 return c1.getName().compareTo(c2.getName());  
 }  
 return i;  
 }  
 return 0;  
 }  
 });  
  
 set.add(new Customer("AA", 1003));  
 set.add(new Customer("BB", 1002));  
 set.add(new Customer("GG", 1004));  
 set.add(new Customer("CC", 1001));  
 set.add(new Customer("DD", 1001));  
  
 for (Object str : set) {  
 System.***out***.println(str);  
 }  
}