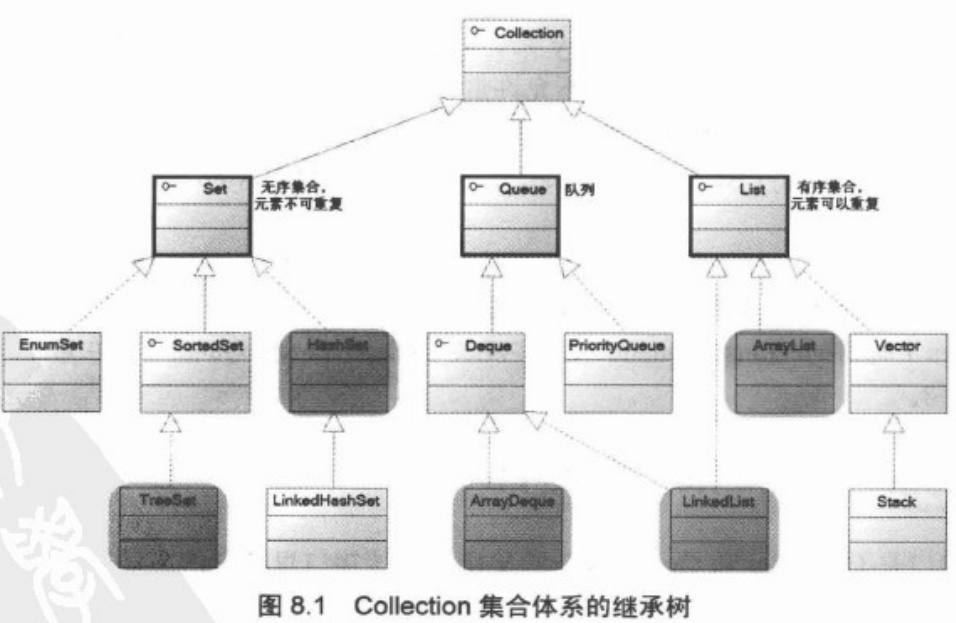
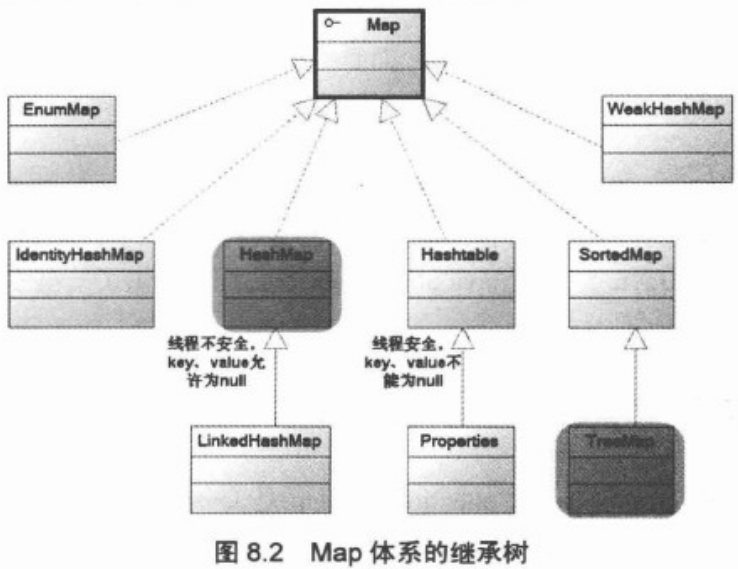
所有的集合类都位于java.util包下，后来为了处理多线程环境下的并发安全问题，java 5 还在java.util.concurrent包下提供了一些多线程支持的集合类。集合类和数组不一样，数组元素既可以是基本类型的值，也可以是对象（实际上保存的是对象的引用）；而集合里只能保存对象（实际上只是保存对象的引用变量，但通常习惯上认为集合里保存的是对象）。

Java的集合类主要由两个接口派生而出：Conllection和Map，Collection和Map是java集合框架的根接口，这两个接口又包含了一些子接口或实现类。







Collection接口是List、Set和Queue接口的父接口，该接口里定义的方法既可用于操作Set集合，也可用于操作List和Queue集合。

Collection collection = **new** ArrayList();

collection.add("hello world");

collection.add(10);

System.*out*.println("collection size = " + collection.size());

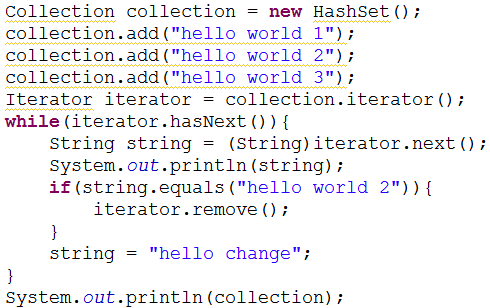


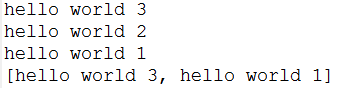
注意：在普通情况下，当我们把一个对象“丢进”集合后，集合会忘记这个对象的类型---也就是说，系统把所有的集合元素都当成Object类的实例进行处理，从JDK 1.5以后，这种状态得到了改进，可以使用泛型类限制集合里的元素类型，并让集合记住所有集合元素的类型。

**遍历集合元素的两种方法**

（1）使用Iterator接口遍历集合元素

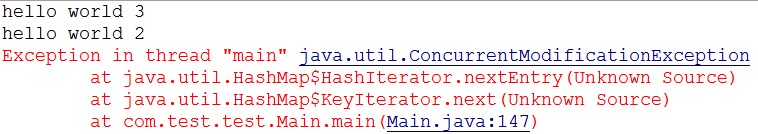
Interator主要用于遍历（即迭代访问）Collection集合中的元素，Iterator对象也被称为迭代器





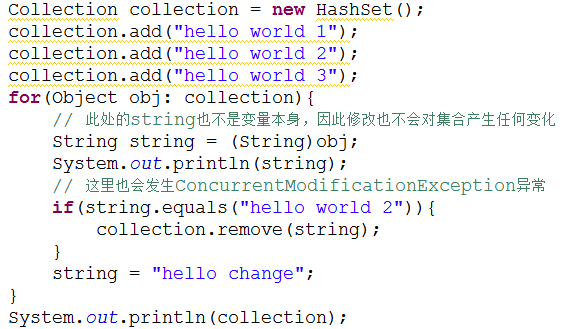
从上面的代码可以看出，Iterator仅用于遍历集合，Iterator本身并不提供盛装对象的能力，如果需要创建Iterator对象，则必须有一个呗迭代的集合，没有集合的Iterator是无本之木，没有存在的价值，Iterator必须依附于Collection对象。同时，对迭代变量string变量进行赋值也没有对集合中的元素有任何变化，这就得到了一个结论：当使用Iterator对集合元素进行迭代时，Iterator并不是把集合元素本身传给了迭代变量，而是把集合元素的值传给了迭代变量，所以修改迭代变量的值对集合元素本身没有任何影响。

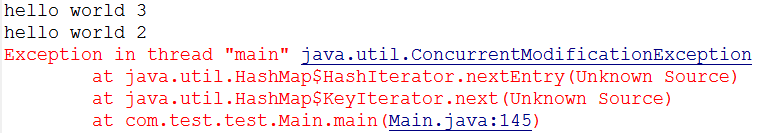
当使用Iterator迭代器访问Collection集合元素时，Collection集合里的元素不能被改变，只能通过Iterator的remove方法删除上一次next方法返回的集合元素才可以；否则会引发java.util.ConcurrentModificationException异常。比如将上面的iterator.remove方法修改为collection.remove(string)时：



Iterator迭代器采用的是快速失败机制，一旦在迭代过程中检测到该集合已经被修改（通常是程序中的其他线程修改），程序立即引发ConcurrentModificationException异常，而不是现实修改后的结果，这样可以避免共享资源而引发的潜在问题。

（2）使用foreach循环遍历集合元素





Foreach循环中的迭代变量也不是集合元素本身，系统知识依次把集合元素的值赋给迭代变量，因此foreach循环中修改迭代变量的值也没有任何实际意义，同样，当使用foreach循环迭代访问集合元素时，该集合也不能被改变，否则将依法ConcurrentModifacationException异常。

**Set集合**

实际上set就是Collection，只是set不允许包含重复元素

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，则添加操作失败，add方法返回false，且新元素不会被加入。Set判断两个对象相同不是使用==运算符，而是根据equals方法。也就是说，只要两个对象用equals方法比较返回true，set就不会接受这两个对象；反之，只要两个对象用equals方法比较返回false，set就会接受这两个对象（甚至这两个对象是同一个对象，set也可把他们当成两个对象处理）

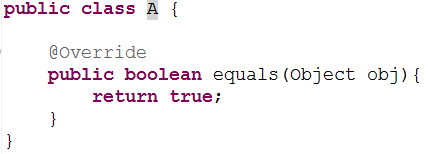
（1）HashSet类

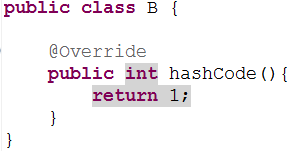
HashSet是Set接口的典型实现，它按Hash算法来存储集合中的元素，因此具有很好的存取和查找性能。

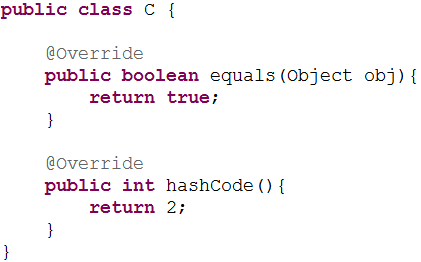
HashSet具有以下特点：

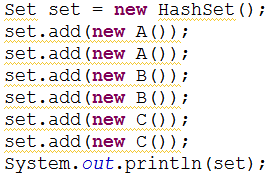
1. 不能保证元素的排列顺序，顺序有可能发生变化
2. HashSet不是同步的，不许通过代码来保证其同步
3. 集合元素值可以是null

当向HashSet集合冲存入一个元素时，HashSet会调用该对象的hashCode方法来得到该对象的hashCode值，然后根据该HashCode值决定该对象在HashSet中的存储位置，如果有两个元素通过equals方法比较返回true吗，但它们的hashCode值不相等，HashSet将会把它们存储在不同的位置，依然可以添加成功。简单地说，HashSet集合盘算两个元素相等的标准是两个对象通过equals方法比较相等，并且两个对象的hashCode方法返回值也相等。











注意：如果需要把某个类的对象保存到HashSet集合中，重写这个类的equals方法和hashCode方法时，应该尽量保证两个对象通过equals方法比较返回true时，它们的hashCode方法返回值也相等。

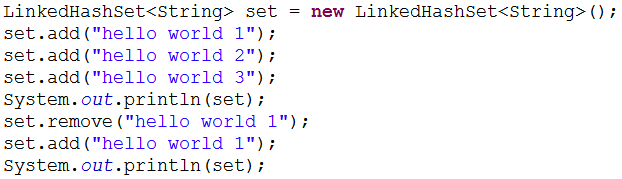
hashSet中每个能存储元素的“槽位”（slot）通常称为“桶”（bucket），如果有多个元素hashCode的值相同，但它们通过equals方法比较返回false，就需要在一个“桶”里放多个元素，这样会导致性能下降。

重写hashCode方法应该遵守的基本规则：

1. 在程序运行过程中，同一个对象多次调用hashCode方法应该返回相同的值
2. 当两个对象通过equals方法比较返回true时，这两个对象的hashCode方法应该返回相等的值
3. 对象中用过equals方法比较标准的Field，都应该用来计算hashCode的值

（2）LinkedHashSet类

HashSet还有一个子类LinkedHashSet，linkedHashSet集合也是根据元素的hashCode值类决定元素的存储位置，但它同时使用链表维护元素的次序，这样使得元素看起来是以插入的顺序保存的，也就是说，当遍历LinkedHashSet集合里的元素时，linkedHashSet将会按元素添加顺序来访问集合里的元素。

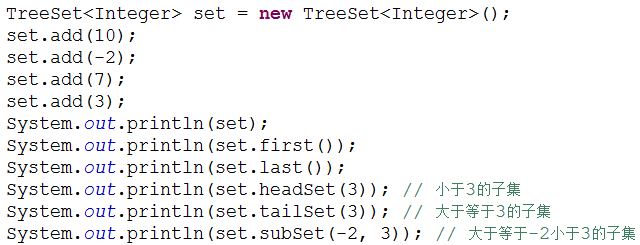


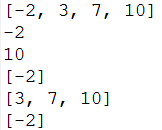


注意：虽然LinkedHashSet使用了链表记录集合元素的添加顺序，但LinkedHashSet依然是HashSet，因此它依然不允许集合元素重复。

（3）TreeSet类

TreeSet是SortedSet接口的实现类，正如SortedSet名字所暗示的，TreeSet可以确保集合元素处于排序状态。





根据上面的运行结果可以看出，TreeSet并不是根据元素的插入顺序进行排序的，而是根据元素实际值的大小来进行排序的。

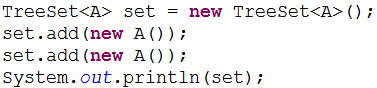
与HashSet集合采用hash算法来决定元素的存储位置不同，TreeSet采用红黑树的数据结构来存储集合元素，那么TreeSet进行排序的规则是怎样的呢？TreeSet支持两种排序方法：自然排序和定制排序

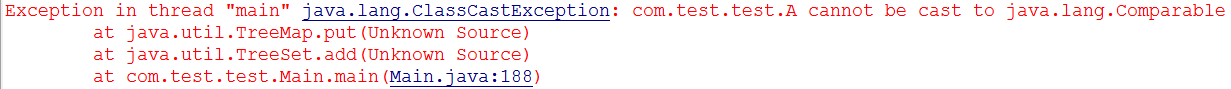
1）自然排序

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，这种方式就是自然排序。

Java提供了一个Comparable接口，该接口里定义了一个compareTo(Object obj)方法。例如obj1.compareTo(obj2)，如果该方法返回0，则表明这两个对象相等，如果该方法返回一个正整数，则表明obj1大于obj2，否则表明obj1小于obj2

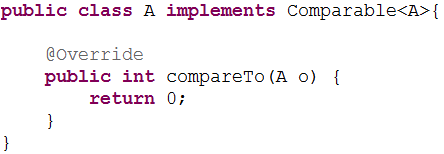
如果试图把一个对象添加到TreeSet时，则该对象的类必须实现Comparable接口，否则程序将会抛出异常。





上面的程序中试图向TreeSet集合中添加两个A，添加第一个对象时候，TreeSet里没有任何元素，所以不会出现任何问题，当添加第二个A对象时，TreeSet就会调用对象的compareTo方法与几何中的其他元素进行比较---如果其对应的类没有实现Compareable接口，则会引发ClassCastException异常。

注意：向TreeSet集合中添加元素时，只有第一个元素无须实现Comparable接口，后面添加的所有元素都必须实现Comparable接口，放然这也不是一种好做法，因为当视图从TreeSet中取出数据时，依然会引发ClassCastException异常。修改之后：





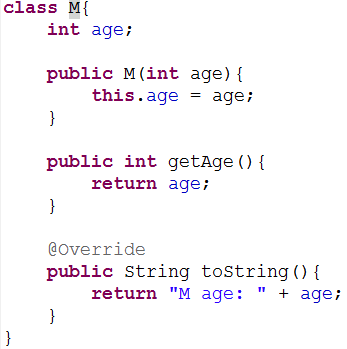
如果将compareTo的函数改为return 1，则运行结果为：

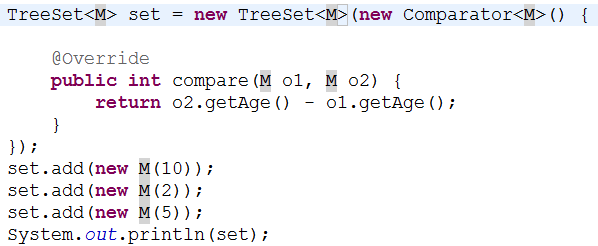


对于TreeSet集合而言，判断两个对象是否相等的唯一标准是：两个对象通过compareTo方法比较是否返回0，如果返回0则认为它们相等，否则认为它们不相等。

2）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，例如降序排序，则可以通过Comparator接口的帮助，该几口中包含一个int compare(T o1, T o2)的方法，如果该方法返回正整数，则表明o1大于o2；如果返回0，则表示两者相等，否则表明o1小于o2







上面的代码中我们创建了一个Comparator接口的匿名内部类对象，该对象负责set集合的排序，所以当我们把M对象添加到set集合中时，无需M类实现Comparable接口，因为此时TreeSet无需通过M对象本身来比较大小，而是由与TreeSet关联的Comparator对象来负责集合元素的排序。

（4）EnumSet类

EnumSet在内部以位向量的形式存储，这种存储形式非常紧凑、高效，因此EnumSet对象占用内存很小，而且运行效率很高。

**各种Set实现类的性能分析**

HashSet和TreeSet是Set的两个典型实现，到底应该如何选择？HashSet的性能总是比TreeSet好（特别是最常用的添加、查询元素等操作），因为TreeSet需要额外的红黑树算法来维护集合元素的次序，只有当需要一个保持排序的Set时，才应该使用TreeSet，否则都应该使用HashSet

HashSet还有一个子类：LinkedHashSet，对于普通的插入、删除操作，LinkedHashSet比HashSet要略微慢一点，这是由于维护链表所带来的饿饿爱开销造成的，不过因为有了链表。遍历LinkedHashSet会更快。

EnumSet是所有Set实现类中性能最好的，但它只能保存同一个枚举类的枚举值作为集合元素

注意：Set的三个实现类HashSet、TreeSet和EnumSet都是线程不安全的，如果有多个线程同时访问一个Set集合，并且有超过一个线程修改了该Set集合，则必须手动保证该Set集合的同步性，通常可以通过Collections工具类的synchronizedSortedSet方法来“包装”该Set集合，此操作最好的创建时进行，以防止对Set集合的意外非同步访问。

**List集合**