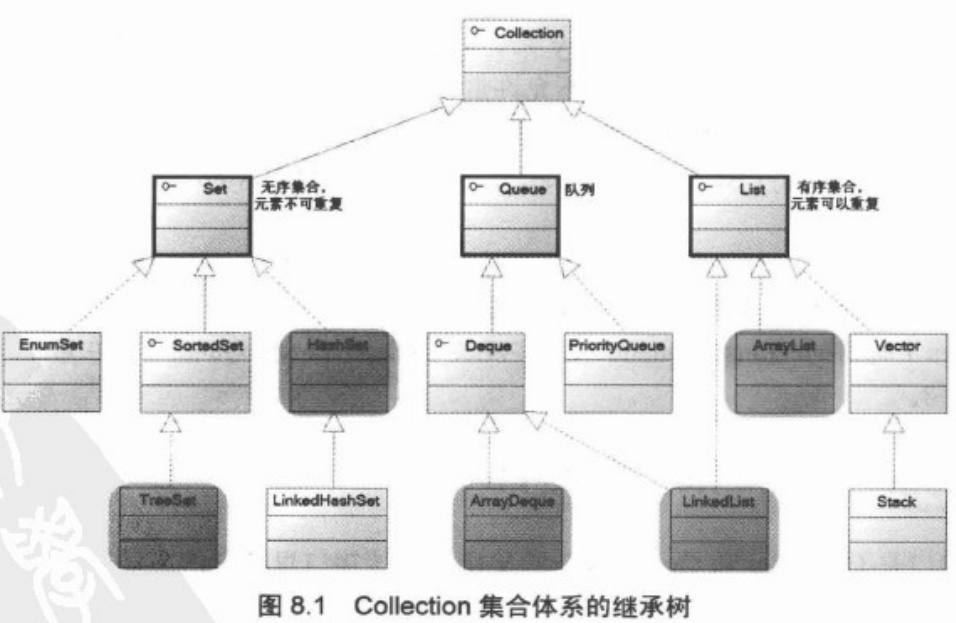
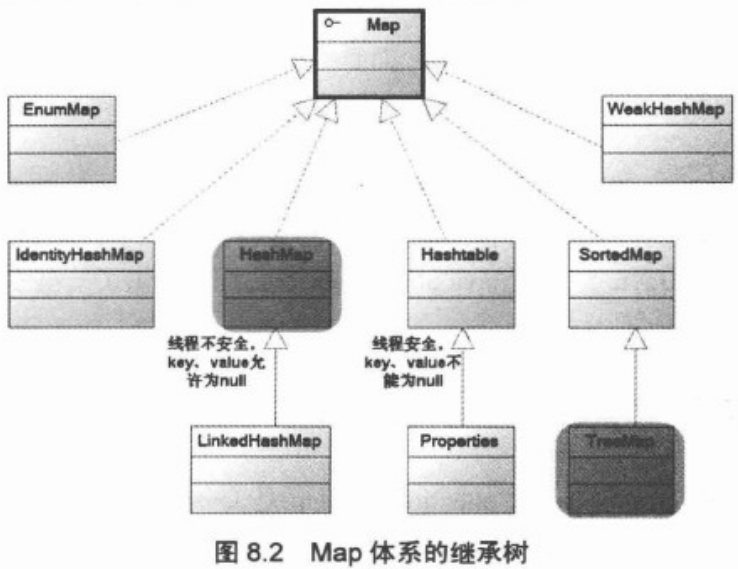
所有的集合类都位于java.util包下，后来为了处理多线程环境下的并发安全问题，java 5 还在java.util.concurrent包下提供了一些多线程支持的集合类。集合类和数组不一样，数组元素既可以是基本类型的值，也可以是对象（实际上保存的是对象的引用）；而集合里只能保存对象（实际上只是保存对象的引用变量，但通常习惯上认为集合里保存的是对象）。

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());

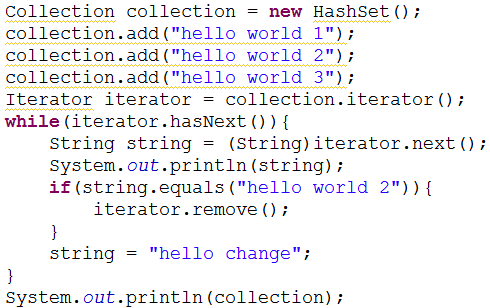


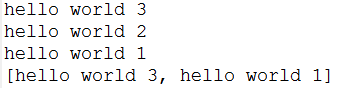
注意：在普通情况下，当我们把一个对象“丢进”集合后，集合会忘记这个对象的类型---也就是说，系统把所有的集合元素都当成Object类的实例进行处理，从JDK 1.5以后，这种状态得到了改进，可以使用泛型类限制集合里的元素类型，并让集合记住所有集合元素的类型。

**遍历集合元素的两种方法**

（1）使用Iterator接口遍历集合元素

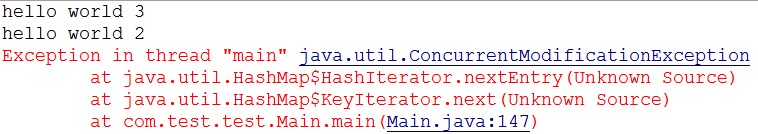
Interator主要用于遍历（即迭代访问）Collection集合中的元素，Iterator对象也被称为迭代器





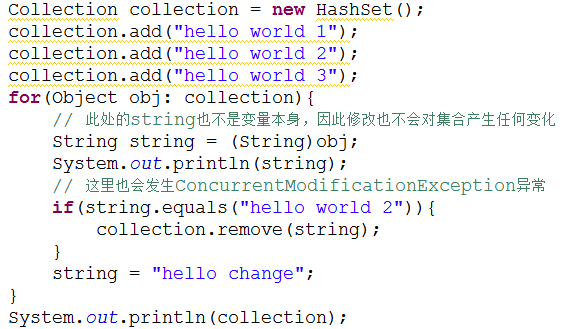
从上面的代码可以看出，Iterator仅用于遍历集合，Iterator本身并不提供盛装对象的能力，如果需要创建Iterator对象，则必须有一个呗迭代的集合，没有集合的Iterator是无本之木，没有存在的价值，Iterator必须依附于Collection对象。同时，对迭代变量string变量进行赋值也没有对集合中的元素有任何变化，这就得到了一个结论：当使用Iterator对集合元素进行迭代时，Iterator并不是把集合元素本身传给了迭代变量，而是把集合元素的值传给了迭代变量，所以修改迭代变量的值对集合元素本身没有任何影响。

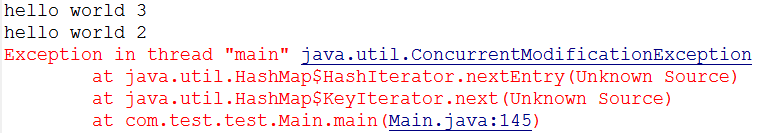
当使用Iterator迭代器访问Collection集合元素时，Collection集合里的元素不能被改变，只能通过Iterator的remove方法删除上一次next方法返回的集合元素才可以；否则会引发java.util.ConcurrentModificationException异常。比如将上面的iterator.remove方法修改为collection.remove(string)时：



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）

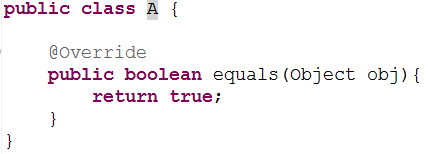
（1）HashSet类

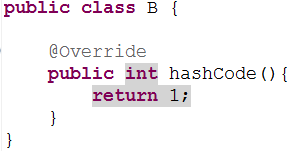
HashSet是Set接口的典型实现，它按Hash算法来存储集合中的元素，因此具有很好的存取和查找性能。

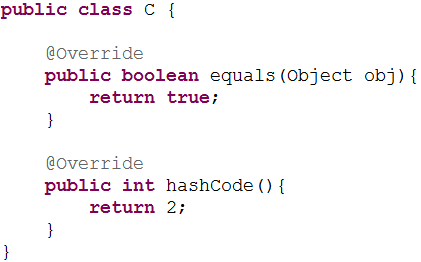
HashSet具有以下特点：

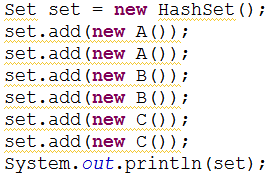
1. 不能保证元素的排列顺序，顺序有可能发生变化
2. HashSet不是同步的，不许通过代码来保证其同步
3. 集合元素值可以是null

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。

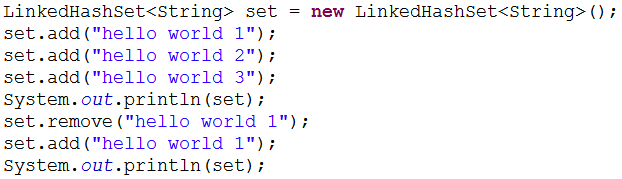
hashSet中每个能存储元素的“槽位”（slot）通常称为“桶”（bucket），如果有多个元素hashCode的值相同，但它们通过equals方法比较返回false，就需要在一个“桶”里放多个元素，这样会导致性能下降。

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。

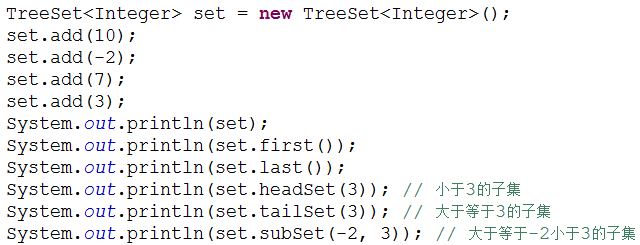


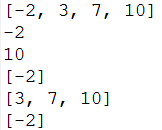


注意：虽然LinkedHashSet使用了链表记录集合元素的添加顺序，但LinkedHashSet依然是HashSet，因此它依然不允许集合元素重复。

（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。

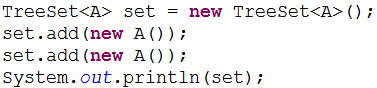
与HashSet集合采用hash算法来决定元素的存储位置不同，TreeSet采用红黑树的数据结构来存储集合元素，那么TreeSet进行排序的规则是怎样的呢？TreeSet支持两种排序方法：自然排序和定制排序

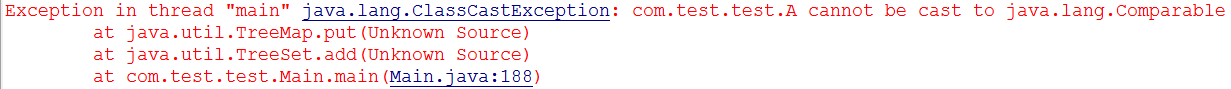
1）自然排序

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

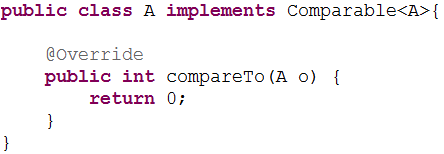
如果试图把一个对象添加到TreeSet时，则该对象的类必须实现Comparable接口，否则程序将会抛出异常。





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：





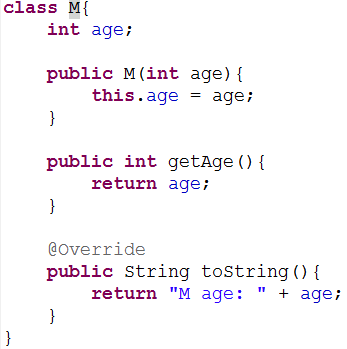
如果将compareTo的函数改为return 1，则运行结果为：

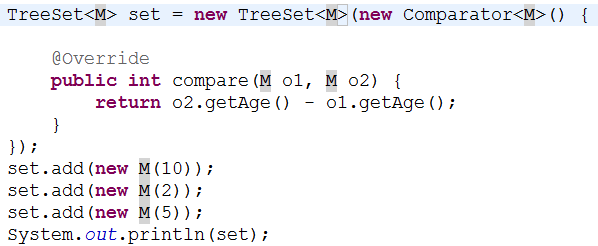


对于TreeSet集合而言，判断两个对象是否相等的唯一标准是：两个对象通过compareTo方法比较是否返回0，如果返回0则认为它们相等，否则认为它们不相等。

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

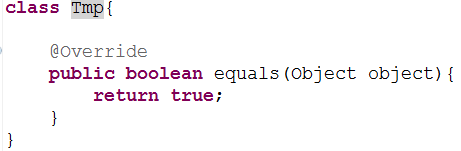
HashSet还有一个子类：LinkedHashSet，对于普通的插入、删除操作，LinkedHashSet比HashSet要略微慢一点，这是由于维护链表所带来的饿饿爱开销造成的，不过因为有了链表。遍历LinkedHashSet会更快。

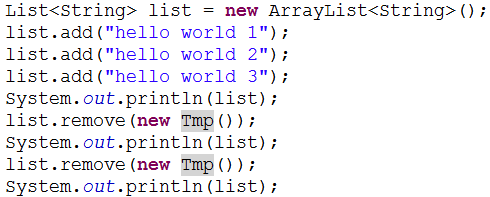
EnumSet是所有Set实现类中性能最好的，但它只能保存同一个枚举类的枚举值作为集合元素

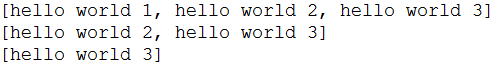
注意：Set的三个实现类HashSet、TreeSet和EnumSet都是线程不安全的，如果有多个线程同时访问一个Set集合，并且有超过一个线程修改了该Set集合，则必须手动保证该Set集合的同步性，通常可以通过Collections工具类的synchronizedSortedSet方法来“包装”该Set集合，此操作最好的创建时进行，以防止对Set集合的意外非同步访问。

**List集合**

List判断两个对象相等只要通过equals方法比较返回true即可，例如：







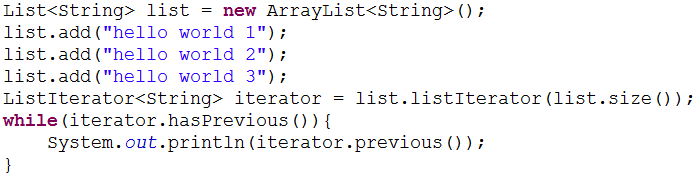
与Set只提供了一个iterator方法不同，List还额外提供了一个listIterator方法，该方法返回一个ListIterator对象，ListIterator接口继承了Iterator几口，提供了专门操作List的方法，ListIterator接口再Iterator接口基础上增加了如下方法：

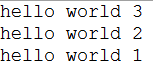
Boolean hasPrevious()

Object previous()

Void add()

我们可以发现ListIterator增加了向前迭代的功能，而且ListIterator还可以通过add方法向List集合中添加元素（Iterator只能删除元素）





（1）ArrayList与Vector类

ArrayList与Vector类都是基于数组实现的List类，所以ArrayList与Vector类封装了一个动态的允许再分配的Object[]数组。

ArrayList与Vector几乎完全相同，但是Vector是一个古老的集合，那时候java还没有提供系统的集合框架，所以Vector中提供了一些方法名很长的方法，在jdk 1.2以后，java提供了系统的集合框架，就将Vector改为实现List几口，从而导致Vector中有一些功能重复的方法。实际上，Vector有很多缺点，通常尽量少用Vector实现类。

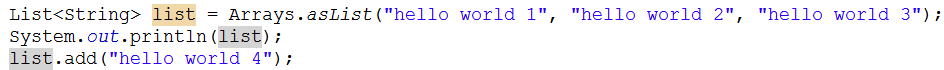
除此之外，ArrayList与Vector的一个显著区别是：ArrayList是线程不安全的，Vector是线程安全的，Vector的性能比ArrayList要低，其实比较源码会发现，Vector中的很多方法都加上了synchronized关键字。实际上，即使需要保证List集合线程安全，也同样不推荐使用Vector实现类，后面会介绍一个Collections工具类，它可以将一个ArrayList变成线程安全的。

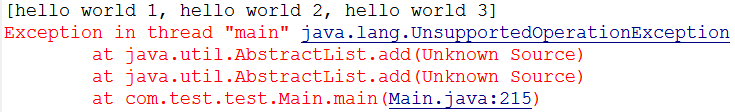
注意：由于Stack继承了Vector，因此它也是一个非常古老的集合类，它是线程安全的，性能比较差，因此现在的程序中一般较少使用Stack类，如果程序需要使用“栈”这种数据结构，则可以考虑使用LinkedList。LinkedList也是List的实现类，它是一个基于链表实现的List类，对于顺序访问集合中的元素进行了优化，特别是插入语删除。LinkedList既实现了List接口，也实现了Deque接口，由于实现了Deque接口，因此可以作为栈来使用。

（2）固定长度的List

在数组操作类Arrays中提供了asList(Object… a)方法，该方法可以将一个数组或指定个数的对象转换成一个List集合，这个List既不是ArrayList实现类的实例，也不是Vector实现类的实例，而是Arrays的内部类ArrayList的实例（已经验证）

Arrays.ArrayList是一个固定长度的List集合，程序只能遍历访问该集合里的元素，不能增加、删除该集合中的元素



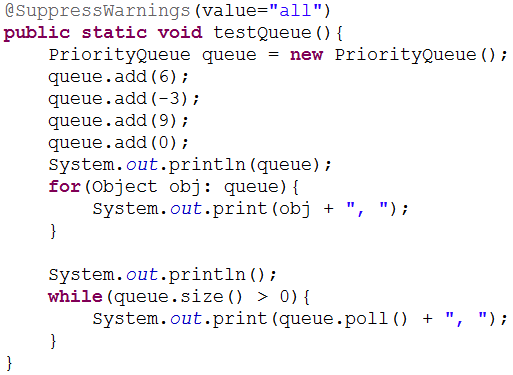


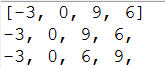
**Queue集合**

Queue接口有一个PriorityQueue实现类，还有一个Deque接口，Deque代表一个“双端队列”，双端队列可以同时从两端来添加、删除元素，因此Deque的实现类既可当成队列使用，也可当做栈使用。Java为Deque提供了ArrayDeque和LinkedList两个实现类。

（1）PriorityQueue实现类

PriorityQueue保存队列元素的顺序并不是按加入队列的顺序，而是按队列元素的大小进行重新排序，因此当调用peek方法或者poll方法取出队列中的元素时，并不是取出最先进入队列的元素，而是取出队列中最小的元素。





运行上面的程序直接输出PriorityQueue集合时，可能看到该队列中的元素并没有很多好地按照大小进行排序，但这只是受到PriorityQueue的toString方法的返回值的影响，实际上，程序多次调用PriorityQueue集合对象的pool方法，即可看到元素按照从小到大的顺序“移出队列”PriorityQueue队列对元素的要求与TreeSet元素的要求基本一致，也可以分为自然排序和定制排序。

（2）Deque接口与ArrayDeque实现类

Deque接口是Queue接口的子接口，它代表一个双端队列，该类中海实现了pop和push两个方法，因此Deque不仅可以当成双端队列使用，而且可以被当成栈类使用。

（3）LinkedList类

LinkedList与ArrayList、ArrayDeque的实现机制完全不同，它内部以链表的形式来保存集合中的元素，因此随机访问集合元素时性能较差，但在插入、删除元素时性能非常出色

**Map**

如果把Map里的所有key放在一起来看，就组成了一个Set集合，实际上Map确实包含了一个keySet方法用于返回Map里所有key组成的Set集合。不仅如此，Map里key集和Set集元素的存储形式也很像，Map子类和Set子类在名字上也惊人地相似，比如Set接口下有HashSet、LinkedHashSet、SortedSet、TreeSet和EnumSet等子接口和实现类，而Map接口下则由HashMap、LinkedHashMap、SortedMap、TreeMap、EnumMap等子接口和实现类，正如他们名字所暗示的，Map的这些实现类和子接口中key集的存储形式和对应Set集合中元素的存储形式完全相同。

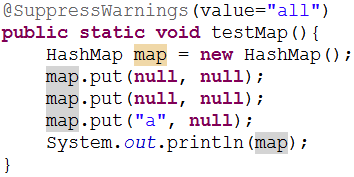
（1）HashMap和Hashtable实现类

HashMap和Hashtable都是Map接口的典型实现类，他们之间的关系完全类似于ArrayList和Vector的关系：Hashtable是一个古老的Map实现类。

Hashtable和HashMap存在两点典型区别：

1）Hashtable是一个线程安全的Map实现，但是HashMap是线程不安全的实现，所以HashMap比Hashtable的性能高一点

2）Hashtable不允许使用null作为key和value，但HashMap可以



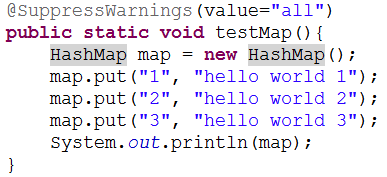


为了能够成功地在HashMap、Hashtable中存储、获取对象，用作key的对象必须实现hashCode方法和equals方法。与HashSet集合不能保证元素的顺序一样，HashMap、Hashtable也不能保证其中key-value对的顺序。类似于HashSet，HashMap、Hashtable判断两个key相等的标准也是：两个key通过equals方法比较返回true，两个key的hashCode值也相等。

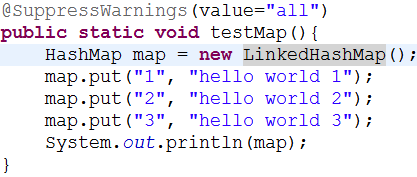
除此之外，HashMap、Hashtable中判断两个value相等的标准更简单：只要两个对象通过equals方法比较返回true即可

（2）LinkedHashMap实现类

HashSet有一个子类是LinkedHashSet，HashMap也有一个LinkedHashMap子类：LinkedHashMap也使用双向链表维护key-value对的次序（其实只需要考虑key的次序），该链表负责维护Map的迭代顺序，迭代顺序与key-value对的插入顺序一致



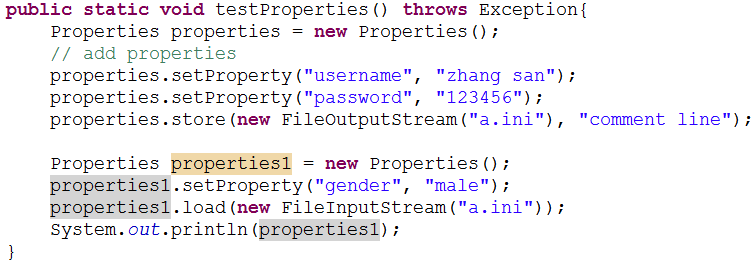




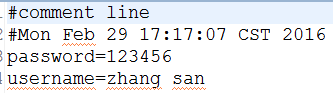


**使用Properties读写属性文件**

Properties类是Hashtable类的子类，该对象在处理属性文件时特别方便。Properties可以把Map对象中的key-value对写入属性文件中，也可以把属性文件中的“属性名=属性值”加载到Map对象中。



文件a.ini内容为：



控制台输出为：



（3）SortedMap接口和TreeMap实现类

正如Set接口派生出SortedSet子接口，SortedSet接口又隔TreeSet实现类一样，Map接口也派生出一个SortedMap子接口，SortedMap接口也有一个TreeMap实现类。

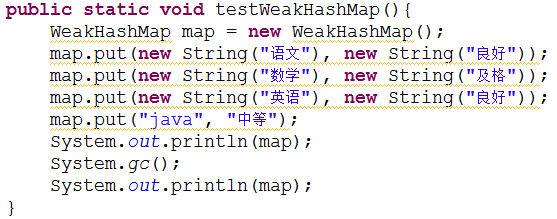
TreeMap就是一个红黑树数据结构，每个key-value对即作为红黑树的一个节点。TreeMap存储key-value对时，需要根据key对节点进行排序，TreeMap也有自然排序和定制排序两种方式。类似于TreeSet中判断两个元素相等的标准，TreeMap中判断两个key相等的标准是：两个key通过compareTo方法返回0则认为相等。

**注意：Set和Map的关系十分密切，java源码就是先实现了HashMap、TreeMap等集合，然后通过包装一个所有的value都为null的Map集合实现了Set集合类**

（4）WeakHashMap类

WeakHashMap与HashMap的用法很相似，与HashMap的区别在于，HashMap的key保留了对实际对象的强引用，这意味着只要该HashMap对象不被销毁，该HashMap的所有key所引用的对象就不会被垃圾回收，HashMap也不会自动删除这些key所对应的key-value对；但WeakHashMap的key只保留了对实际对象的弱引用，这意味着如果WeakHashMap对象的key所引用的对象没有被其他强引用变量所引用，则这些key所引用的对象可能被垃圾回收，WeakHashMap也可能自动删除这些key所对应的key-value

WeakHashMap中的每个key对象只持有对实际对象的弱引用，因此，当垃圾回收了该key所对应的实际对象之后，WeakHashMap会自动删除该key对象的key-value对。

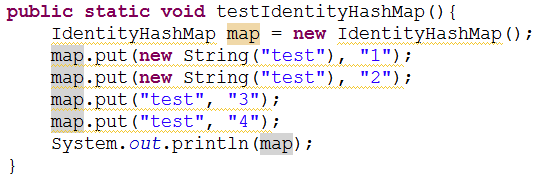




从上面运行结果可以看出，当系统进行垃圾回收时，删除了WeakHashMap对象的前三个key-value对，这是因为添加前三个key-value对时，这三个key都是匿名的字符串对象，WeakHashMap只保留了对他们的弱 引用，这样垃圾回收时会自动删除这三个key-value对，第四组的key-value中的key是一个字符串常量，系统会自动保留对该字符串对象的强引用，所以垃圾回首时不会回收它。

（5）IdentityHashMap实现类

这个Map实现类的实现机制与HashMap基本相似，但是它在处理两个key相等时比较独特，在这个Map中，当且仅当两个key严格相等（key1==key2）时，才会认为这两个key相等。对于普通的HashMap而言，只要key1和key2通过equals方法比较返回true，且他们的hashCode值相等即可。

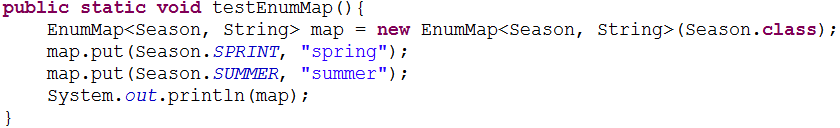




（6）EnumMap

EnumMap是一个与枚举类一起使用的Map实现，EnumMap中所有的key都必须是单个枚举类的枚举值。创建EnumMap时必须显式或隐式指定它对应的枚举类型，EnumMap在内部以数组形式保存，所有这种实现形式非常紧凑、高效。

EnumMap根据key的自然顺序（即枚举值在枚举类中的定义顺序）来维护key-value对的顺序，当程序通过keySet、entrySet、values等方法遍历EnumMap时可以看到这种顺序。它不允许使用null作为key，但允许使用null作为value





**各种Map实现类的性能分析**

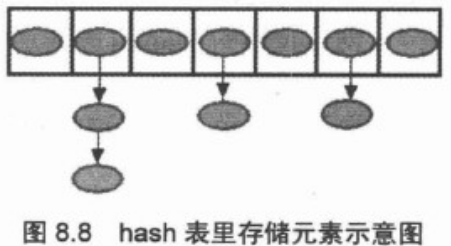
对于Map的常用类而言，HashMap和Hashtable的效率大致相同，因为它们的实现机制几乎完全一样，但HashMap通常比Hashtable要快一点，因为Hashtable需要额外的线程同步控制。

TreeMap通常比HashMap、HashTable要慢（尤其在插入、删除key-value对时更慢），因为TreeMap底层采用红黑树来管理key-value对（红黑树的每个节点就是一个key-value对）

对于一般的场景而言，应该多考虑使用HashMap，因为HashMap正事快速查询设计的（HashMap底层其实也是采用数组来存数key-value对），但是如果程序需要一个总是排好序的Map时，则可以考虑使用TreeMap。

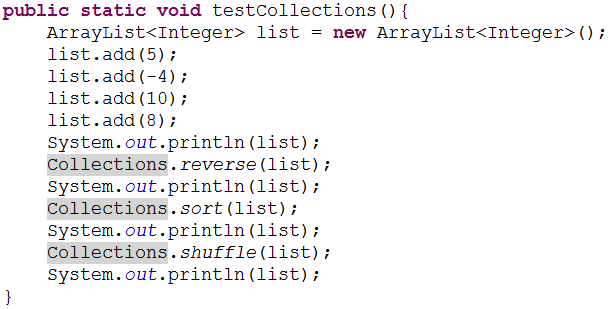
EnumMap的性能最好，但它只能使用同一个美剧类的枚举值多为key

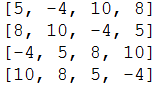
Hash表中可以存储元素的位置被称为“桶”，通常情况下，单个“桶”存储一个元素，在发生冲突的情况下，单个桶会存储多个元素，这些元素以链表形式存储，必须按顺序搜索



**操作集合的工具类：Collections**

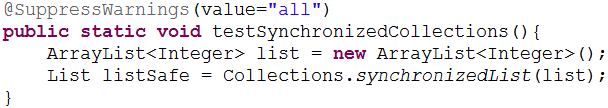
Java提供了一个操作Set、List和Map等集合的工具类：Collections，该工具类中提供了大量方法对集合元素进行排序、查询和修改等操作，还提供了将集合对象设置为不可变，对集合对象实现同步控制等方法。





同步控制

Collections类中提供了多个synchronizedXxx()方法，该方法可以将指定的集合包装成线程同步的集合，从而可以解决多线程并发访问集合时的线程安全问题



**设置不可变集合**

Collections提供了三类方法返回一个不可变的集合：

1）emptyXxx：返回一个空的、不可变的集合，可以为List、Set、Map

2）singletonXxx：返回一个只包含指定对象（只有一个或以项元素）的、不可变的集合对象可以为List、Set、Map

3）unmodifiableXxx：返回指定集合对象的不可变视图，可以为List、Set、Map

