5. Java 集合容器

λ 集合容器概述

ν 什么是集合

集合就是一个放数据的容器，准确的说是放数据对象引用的容器。

ν 集合的特点

* 集合用于存储对象的容器，对象是用来封装数据，对象多了也需要存储集中式管理。
* 和数组对比对象的大小不确定。因为集合是可变长度的。数组需要提前定义大小

ν 集合和数组的区别

* 数组是固定长度的；集合可变长度的。
* 数组可以存储基本数据类型，也可以存储引用数据类型；集合只能存储引用数据类型。
* 数组存储的元素必须是同一个数据类型；集合存储的对象可以是不同数据类型。

ν 使用集合框架的好处

（1）使用集合类可以降低开发成本，而不用自己实现集合类。

（2）因为我们用的集合框架类是经过严格测试的，所以代码质量会得到提高。  
（3）通过使用JDK附带的集合类，可以降低代码维护成本。  
（4）复用性和可操作性

ν 常用的集合类有哪些？

1. Collection接口的子接口包括：Set接口和List接口

2. Map接口的实现类主要有：HashMap、TreeMap、Hashtable、ConcurrentHashMap以及Properties等

3. Set接口的实现类主要有：HashSet、TreeSet、LinkedHashSet等

4. List接口的实现类主要有：ArrayList、LinkedList、Stack以及Vector等

ν List，Set，Map 三者的区别？List、Set、Map 是否继承自 Collection 接口？List、Map、Set 三个接口存取元素时，各有什 么特点？

List与Set都是单列元素的集合，它们有一个共同的父接口Collection。Map不是Collection的子接口，

List

特点： 元素有放入顺序，元素可重复

存元素：多次调用add(Object)方法时，每次加入的对象按先来后到的顺序排序，也可以插队，即调用add(int index,Object)方法，就可以指定当前对象在集合中的存放位置。

取元素：

方法1：Iterator接口取得所有，逐一遍历各个元素。

方法2：调用get(index i)来明确说明取第几个。

for循环，foreach循环，Iterator迭代器迭代。

Set

特点：元素无放入顺序，元素不可重复。

存元素：add方法有一个boolean的返回值，当集合中没有某个元素，此时add方法可成功加入该元素时，则返回true；

当集合含有与某个元素equals相等的元素时，此时add方法无法加入该元素，返回结果为false。

取元素：可以foreach循环，iterator迭代器 迭代

接下来就说Map了。

Map

特点：元素按键值对存储，无放入顺序（键不能重复，值可以重复，非线程安全，高效，支持null；HashTable线程安全，低效，不支持null ）

每次存储时，要存储一对keyalue，不能存储重复的key，这个重复的规则也是按equals比较相等

存元素：存放用put方法:put(obj key,obj value)，每次存储时，要存储一对keyalue，get(Object key)方法根据key获得相应的value。

取元素：Map取出元素需转换为Set，然后进行Iterator迭代器迭代，或转换为Entry对象进行Iterator迭代器迭代

List是用来处理序列的，而Set是用来处理集的。存储的是键值对。

ν集合框架底层数据结构

List

Arraylist： Object数组

Vector： Object数组

LinkedList： 双向链表(JDK1.6之前为循环链表，JDK1.7取消了循环)

Set

HashSet（无序，唯一）: 基于 HashMap 实现的，底层采用 HashMap 来保存元素

LinkedHashSet： LinkedHashSet 继承与 HashSet，并且其内部是通过LinkedHashMap 来实现的。

TreeSet（有序，唯一）： 红黑树(自平衡的排序二叉树。)

Map

HashMap： JDK1.8之前HashMap由数组+链表组成的，数组是HashMap的主体， JDK1.8以后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）时，将链表转化为红黑树，以减少搜索时间

TreeMap: 红黑树（自平衡的排序二叉树）

ν 哪些集合类是线程安全的？

vector：具有同步化机制（线程安全）

statck：堆栈类，先进后出

hashtable：就比hashmap多了个线程安全

enumeration：枚举，相当于迭代器

ν 怎么确保一个集合不能被修改？

用Collections的

unmodifiableCollection（Collection c）

unmodifiableList（List list）

unmodifiableSet（Set set）

unmodifiableMap（Map map）

方法创建只读的集合。

任何修改操作都会抛出UnsupportedOperationException

λ Collection 接口

ν List 接口

υ 迭代器 Iterator 是什么？

Iterator 是可以遍历集合的对象，为各种容器提供了公共的操作接口，隔离对容器的遍历操作和底层实现，从而解耦。

缺点是增加新的集合类需要对应增加新的迭代器类，迭代器类与集合类成对增加。

υ Iterator 怎么使用？有什么特点？

Iterator的使用

（1）Iterator（）要求容器返回一个Iterator。Iterator将准备好返回序列的第一个元素。

（2）使用next（）获得序列中的下一个元素

（3）使用hasNext（）检查序列中是否还有元素。

（4）使用remove（）将迭代器新近返回的元素删除。

Iterator的特点

（1） Iterator遍历集合元素的过程中不允许线程对集合元素进行修改，否则会抛出ConcurrentModificationEception的异常。

（2）Iterator遍历集合元素的过程中可以通过remove方法来移除集合中的元素，删除的是上一次Iterator.next()方法返回的对象。

（3）Iterator必须依附于一个集合类对象而存在，Iterator本身不具有装载数据对象的功能。

（4）next（）方法，该方法通过游标指向的形式返回Iterator下一个元素。

υ 如何边遍历边移除 Collection 中的元素？

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

\*// do something\*

it.remove();

}

υ Iterator 和 ListIterator 有什么区别？

Iterator 可以遍历 Set 和 List 集合，而 ListIterator 只能遍历 List。

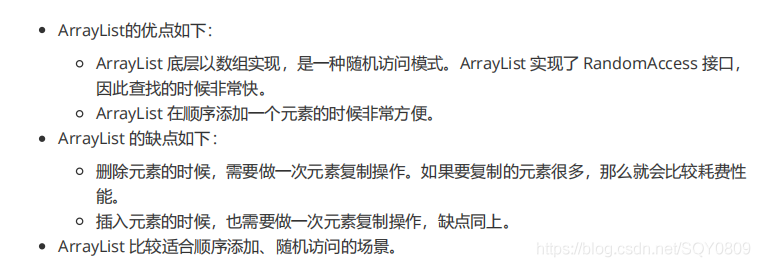
Iterator 只能单向遍历，而 ListIterator 可以双向遍历（向前/后遍历）。

ListIterator 实现 Iterator 接口，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

υ 遍历一个 List 有哪些不同的方式？每种方法的实现原理是什 么？Java 中 List 遍历的最佳实践是什么？



υ 说一下 ArrayList 的优缺点



υ 如何实现数组和 List 之间的转换？

* 数组转 List：使用 Arrays. asList(array) 进行转换。
* List 转数组：使用 List 自带的 toArray() 方法。

υ ArrayList 和 LinkedList 的区别是什么？

数据结构实现：ArrayList 是动态数组的数据结构实现，而 LinkedList 是双向链表的数据结构实现。

随机访问效率：ArrayList 比 LinkedList 在随机访问的时候效率要高，因为 LinkedList 是线性的数据存储方式，所以需要移动指针从前往后依次查找。

增加和删除效率：在非首尾的增加和删除操作，LinkedList 要比 ArrayList 效率要高，因为ArrayList 增删操作要影响数组内的其他数据的下标。

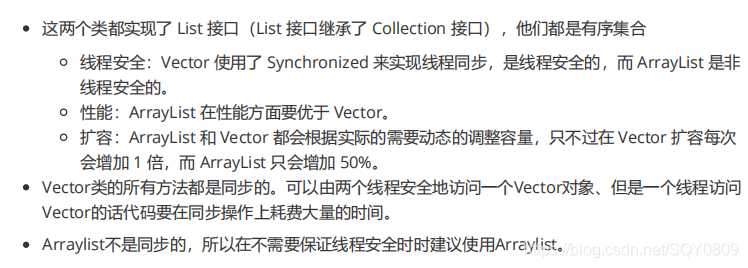
内存空间占用：LinkedList 比 ArrayList 更占内存，因为 LinkedList 的节点除了存储数据，还存储了两个引用，一个指向前一个元素，一个指向后一个元素。

线程安全：ArrayList 和 LinkedList 都是不同步的，也就是不保证线程安全；

综合来说，在需要频繁读取集合中的元素时，更推荐使用 ArrayList，而在插入和删除操作较多时，更推荐使用 LinkedList。

LinkedList 的双向链表也叫双链表，是链表的一种，它的每个数据结点中都有两个指针，分别指向直接后继和直接前驱。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点

υ ArrayList 和 Vector 的区别是什么？



υ 插入数据时，ArrayList、LinkedList、Vector 谁速度较快？阐 述 ArrayList、Vector、LinkedList 的存储性能和特性？

ArrayList和Vector 底层的实现都是使用数组方式存储数据。数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢。

Vector 中的方法由于加了 synchronized 修饰，因此 Vector 是线程安全容器，但性能上较ArrayList差。

LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但插入数据时只需要记录当前项的前后项即可，所以 LinkedList 插入速度较快。

υ 多线程场景下如何使用 ArrayList？

ArrayList 不是线程安全的，如果遇到多线程场景，可以通过 Collections 的 synchronizedList 方法将其转换成线程安全的容器后再使用。例如像下面这样：

List<String> synchronizedList = Collections.synchronizedList(list);

synchronizedList.add("aaa");

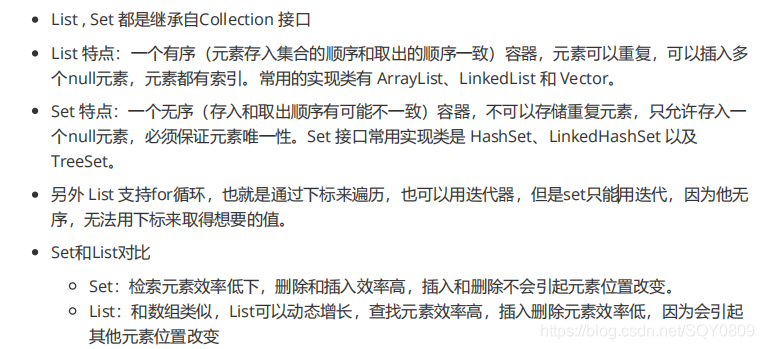
synchronizedList.add("bbb");

for (int i = 0; i < synchronizedList.size(); i++) {

System.out.println(synchronizedList.get(i));

}

υ List 和 Set 的区别



ν Set 接口

υ 说一下 HashSet 的实现原理？

HashSet 是基于 HashMap 实现的，HashSet的值存放于HashMap的key上，HashMap的value统一为present，因此 HashSet 的实现比较简单，相关 HashSet 的操作，基本上都是直接调用底层HashMap 的相关方法来完成，HashSet 不允许重复的值。

υ HashSet 如何检查重复？HashSet 是如何保证数据不可重复 的？

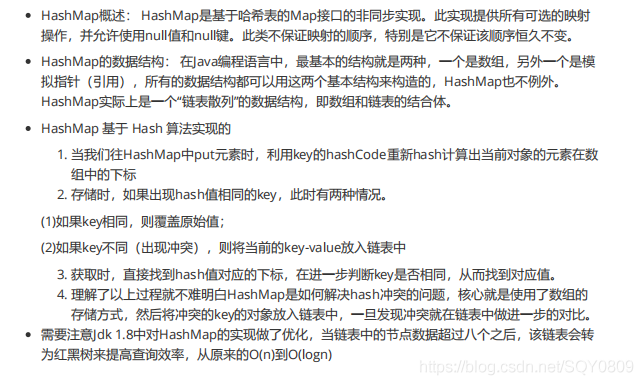
向HashSet 中add ()元素时，判断元素是否存在的依据，不仅要比较hash值，同时还要结合equles 方法比较。

HashSet 中的add ()方法会使用HashMap 的put()方法。

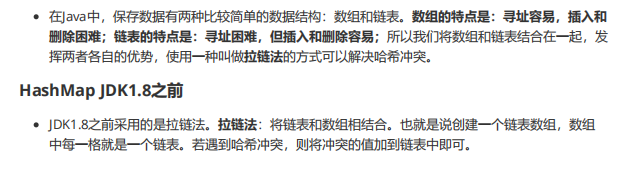
HashMap 的 key 是唯一的，由源码可以看出 HashSet 添加进去的值就是作为HashMap 的key，并且在HashMap中如果K/V相同时，会用新的V覆盖掉旧的V，然后返回旧的V。所以不会重复（HashMap 比较key是否相等是先比较hashcode 再比较equals ）。

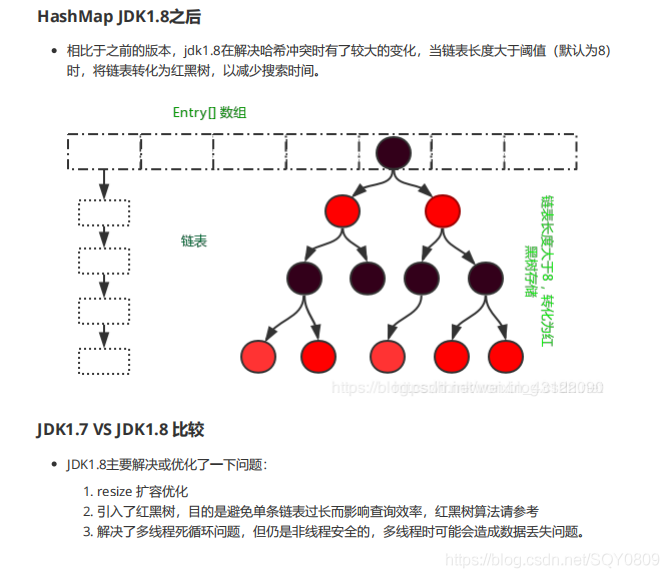
λ Map 接口

ν 说一下 HashMap 的实现原理？



ν HashMap 在 JDK1.7 和 JDK1.8 中有哪些不同？HashMap 的底层 实现

υ JDK1.8 之前

υ JDK1.8 之后υ JDK1.7 VS JDK1.8 比较



ν HashMap 的 put 方法的具体流程？

1. 判断键值对数组table[i]是否为空或为null，否则执行resize()进行扩容；

2. 根据键值key计算hash值得到插入的数组索引i，如果table[i]==null，直接新建节点添加，转向⑥，如果table[i]不为空，转向③；

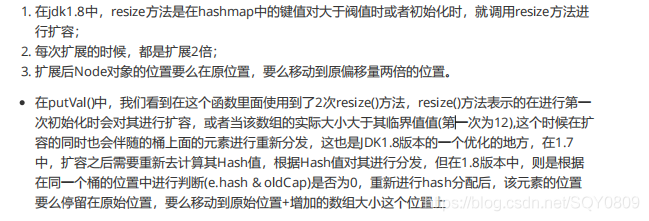
3. 判断table[i]的首个元素是否和key一样，如果相同直接覆盖value，否则转向④，这里的相同指的是hashCode以及equals；

4. 判断table[i] 是否为treeNode，即table[i] 是否是红黑树，如果是红黑树，则直接在树中插入键值对，否则转向5；

5. 遍历table[i]，判断链表长度是否大于8，大于8的话把链表转换为红黑树，在红黑树中执行插入操作，否则进行链表的插入操作；遍历过程中若发现key已经存在直接覆盖value即可；

6. 插入成功后，判断实际存在的键值对数量size是否超多了最大容量threshold，如果超过，进行扩容。

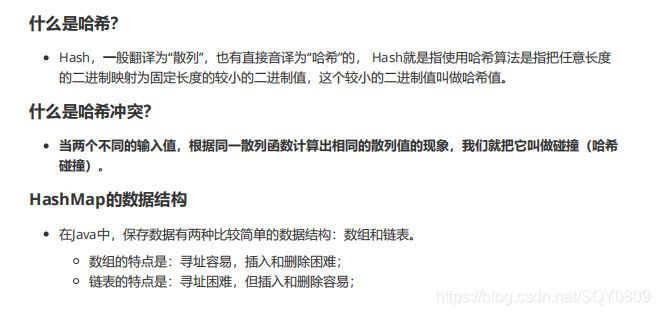
ν HashMap 的扩容操作是怎么实现的？

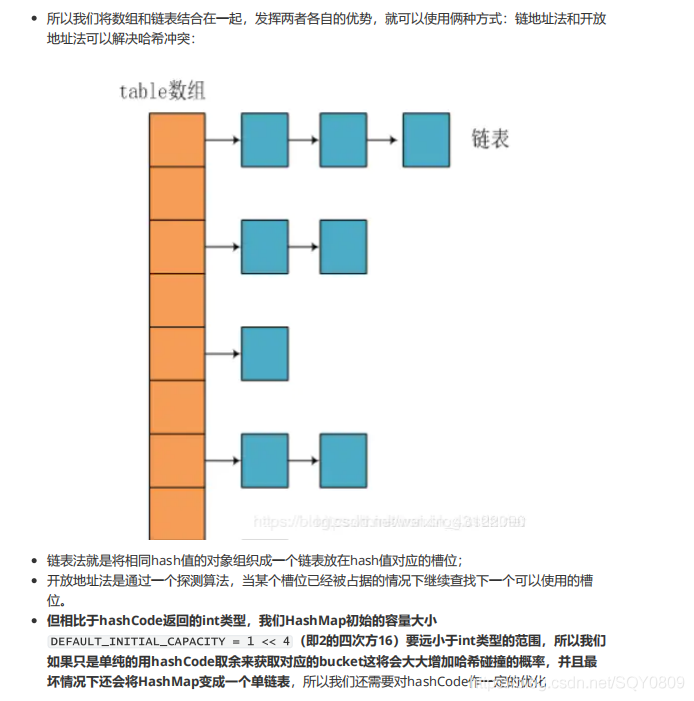


ν HashMap 是怎么解决哈希冲突的？

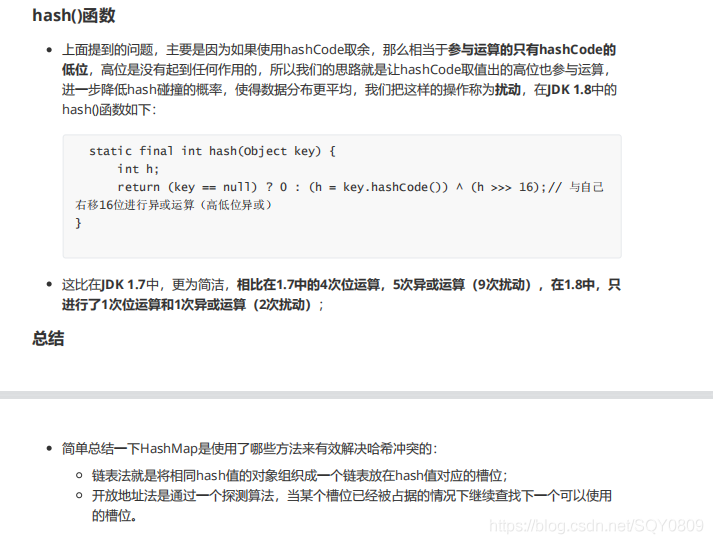
υ 什么是哈希？

υ 什么是哈希冲突？

υ HashMap 的数据结构



υ 总结



ν 能否使用任何类作为 Map 的 key？

可以使用任何类作为 Map 的 key ，然而在使用之前，需要考虑以下几点：

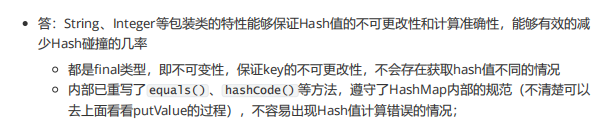
如果类重写了 equals() 方法，也应该重写 hashCode() 方法。

类的所有实例需要遵循与 equals() 和 hashCode() 相关的规则。

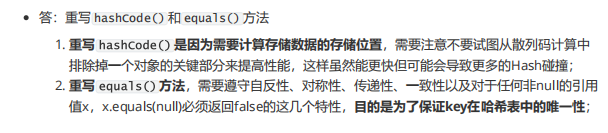
如果一个类没有使用 equals()，不应该在 hashCode() 中使用它。

用户自定义 Key 类最佳实践是使之为不可变的，这样 hashCode() 值可以被缓存起来，拥有更好的性能。不可变的类也可以确保 hashCode() 和 equals() 在未来不会改变，这样就会解决与可变相关的问题了。

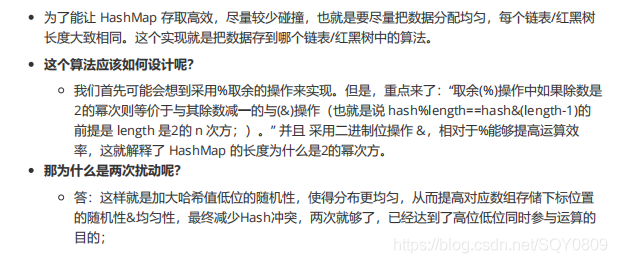
ν 为什么 HashMap 中 String、Integer 这样的包装类适合作为 K？



ν 如果使用 Object 作为 HashMap 的 Key，应该怎么办呢？



ν HashMap 的长度为什么是 2 的幂次方



ν HashMap 与 HashTable 有什么区别？

1. 线程安全： HashMap 是非线程安全的，HashTable 是线程安全的；HashTable 内部的方法基本都经过 synchronized 修饰。（如果你要保证线程安全的话就使用 ConcurrentHashMap ）；

2. 效率： 因为线程安全的问题，HashMap 要比 HashTable 效率高一点。另外，HashTable 基本被淘汰，不要在代码中使用它；（如果你要保证线程安全的话就使用 ConcurrentHashMap ）；

3. 对Null key 和Null value的支持： HashMap 中，null 可以作为键，这样的键只有一个，可以有一个或多个键所对应的值为 null。但是在 HashTable 中 put 进的键值只要有一个 null，直接抛NullPointerException。

4. 初始容量大小和每次扩充容量大小的不同 ：

5. 创建时如果不指定容量初始值，Hashtable 默认的初始大小为11，之后每次扩充，容量变为原来的2n+1。HashMap 默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。

6. 创建时如果给定了容量初始值，那么 Hashtable 会直接使用你给定的大小，而 HashMap 会将其扩充为2的幂次方大小。也就是说 HashMap 总是使用2的幂作为哈希表的大小，后面会介绍到为什么是2的幂次方。

7. 底层数据结构： JDK1.8 以后的 HashMap 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）时，将链表转化为红黑树，以减少搜索时间。Hashtable 没有这样的机制。

8. 推荐使用：在 Hashtable 的类注释可以看到，Hashtable 是保留类不建议使用，推荐在单线程环境下使用 HashMap 替代，如果需要多线程使用则用 ConcurrentHashMap 替代。

ν 如何决定使用 HashMap 还是 TreeMap？

* 对于在Map中插入、删除和定位元素这类操作，HashMap是最好的选择。然而，假如你需要对一个有序的key集合进行遍历，TreeMap是更好的选择。基于你的collection的大小，也许向HashMap中添加元素会更快，将map换为TreeMap进行有序key的遍历。

ν HashMap 和 ConcurrentHashMap 的区别

1. ConcurrentHashMap对整个桶数组进行了分割分段(Segment)，然后在每一个分段上都用lock锁进行保护，相对于HashTable的synchronized锁的粒度更精细了一些，并发性能更好，而HashMap没有锁机制，不是线程安全的。（JDK1.8之后ConcurrentHashMap启用了一种全新的方式实现,利用CAS算法。）

2. HashMap的键值对允许有null，但是ConCurrentHashMap都不允许。

辅助工具类

ν Comparable 和 Comparator 的区别？

comparable接口实际上是出自java.lang包，它有一个 compareTo(Object obj)方法用来排序

comparator接口实际上是出自 java.util 包，它有一个compare(Object obj1, Object obj2)方法用来排序

的Comparator方法或者以两个Comparator来实现歌名排序和歌星名排序，第二种代表我们只能使用两个参数版的Collections.sort().

ν Collection 和 Collections 有什么区别？

java.util.Collection 是一个集合接口（集合类的一个顶级接口）。它提供了对集合对象进行基本操作的通用接口方法。

Collections则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

ν TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具 类中的 sort()方法如何比较元素？

