# 第二十章 线性表、栈、队列和优先队列

## 20.1引言

1.要点提示：为一个特定的任务选择最好的数据结构和算法是开发高性能软件的一个关键。

2.数据结构：数据结构是以某种形式将数据组织在一起的合集。数据结构不仅存储数据，还支持访问和处理数据的操作。（数据存储方式、处理方式）

3.在面向对象思想里，一种数据结构也被认为是一个容器或者容器对象，它是一个能存储其它对象的对象，这里的其它对象常称为数据或者元素。

4.定义一种数据结构从本质上讲就是定义一个类。数据结构类应该使用数据域存储数据，并提供方法支持查找、插入和删除等操作。因此，创建一个数据结构就是创建这个类的一个实例。然后，可以使用这个实例上的方法来操作这个数据结构。

## 20.2合集

1.要点提示：Collection接口为线性表、向量、栈、队列，优先队列以及集合定义了共同的操作。

2.Java合集框架支持以下两种类型的容器：

①一种是为了存储一个元素的合集，简称为“合集”。

②另一种是为了存储键/值对，称为“映射表”。

3.映射表是一个用于是用一个键（key）快速搜索一个元素的高效数据结构。

4.下面介绍合集：（实际就是数据结构的具象化）

Set用于存储一组不重复的元素。

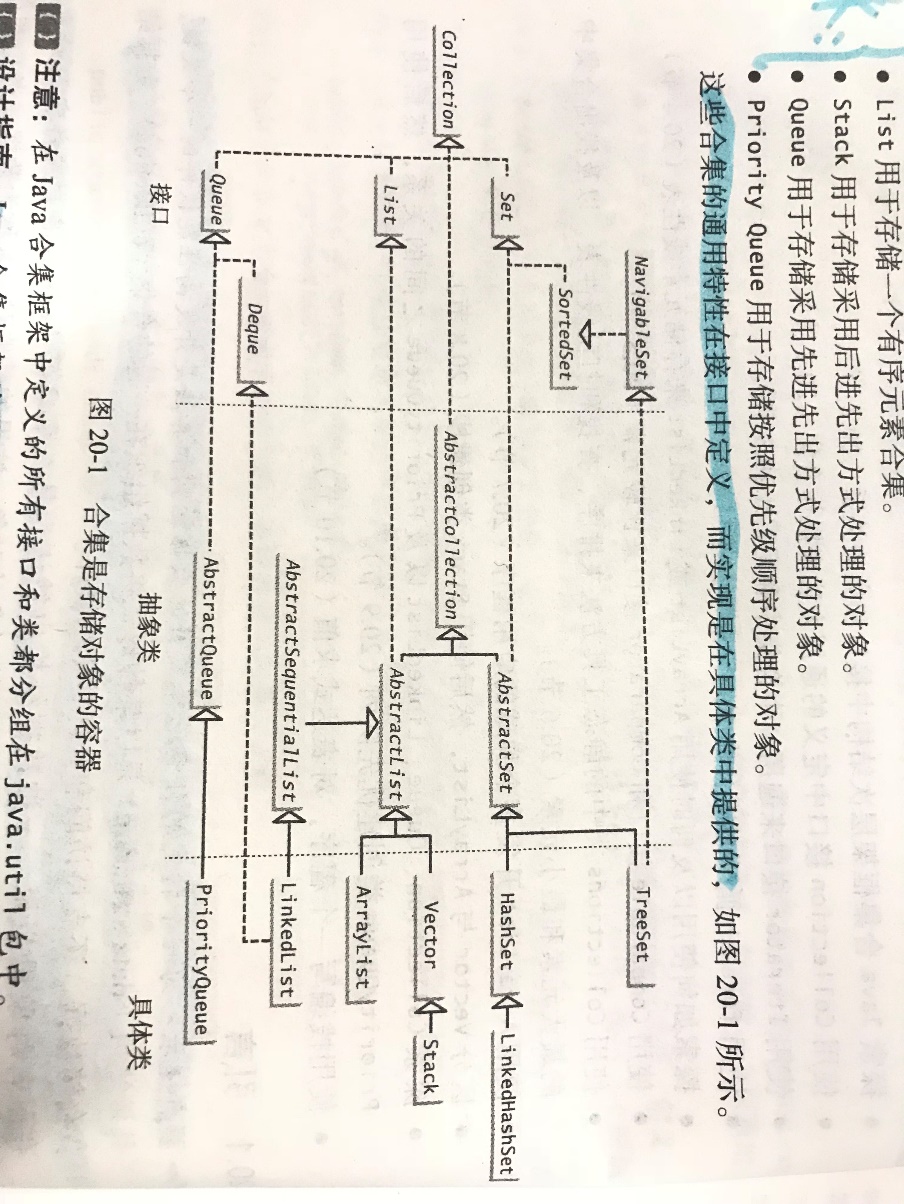
List用于存储一个有序元素合集。

Stack用于存储采用后进先出方式处理的对象。

Queue用于存储采用先进先出方式处理的对象。

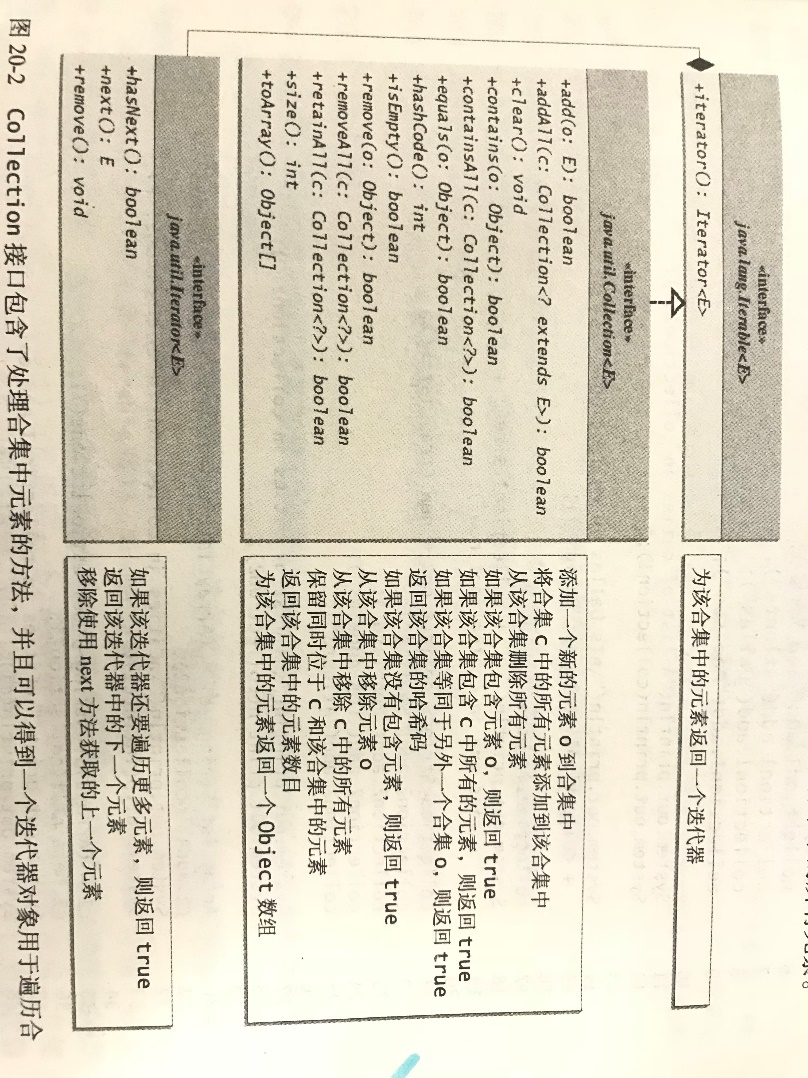
Priority Queue用于存储按照优先级顺序处理的对象。

5.这些合集的通用特性在接口中定义，而实现是在具体类中提供的，如图



Java合集框架的设计是使用接口、抽象类和具体类的一个很好的例子。用接口定义框架。抽象类提供这个接口的部分实现。具体类用具体的数据结构实现这个接口。

6.Collection接口中包含了处理集合中元素的方法，并且可以得到一个迭代器对象用于遍历集合中的元素，如图：



注意：方法addAll、removeAll、retainAll类似于集合上的并、差、交运算。

7.以下程序给出了一个使用定义在Collection接口中方法的示例。



对于该例子来说，我们使用了ArrayList。其实可以使用Collection的任意具体类，如HashSet、LinkedList、Vector以及Stack替代ArrayList来测试这些定义在Collection接口中的方法。

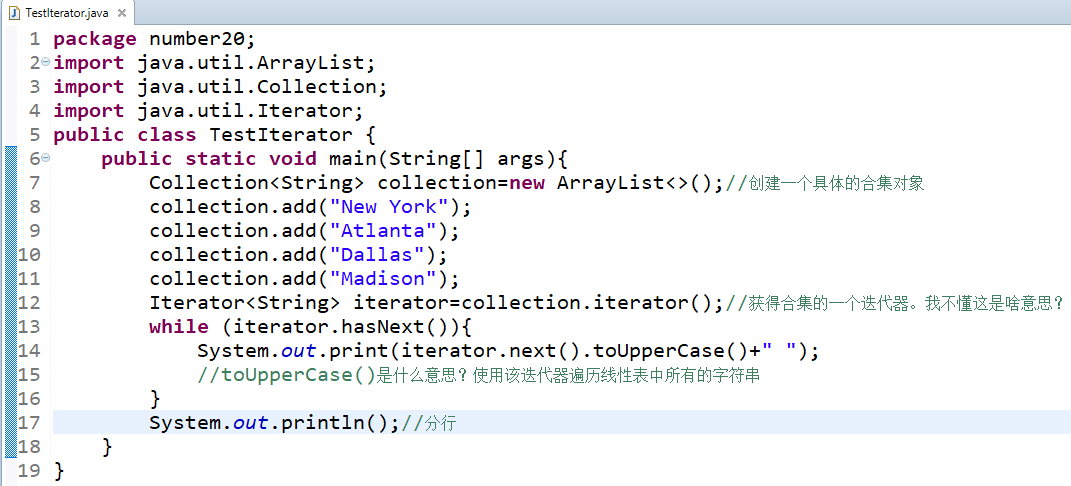
## 20.3迭代器

1.要点提示：每种集合都是可迭代的。可以获得集合的Iterator对象来遍历集合中的所有元素。

2. Iterator是一种经典的设计模式，用于在不需要暴露数据是如何保存在数据结构的细节情况下，来遍历一个数据结构。

3. Iterator接口为遍历各种类型的合集中的元素提供了一种统一的方法。

4.如下图Iterator示例：



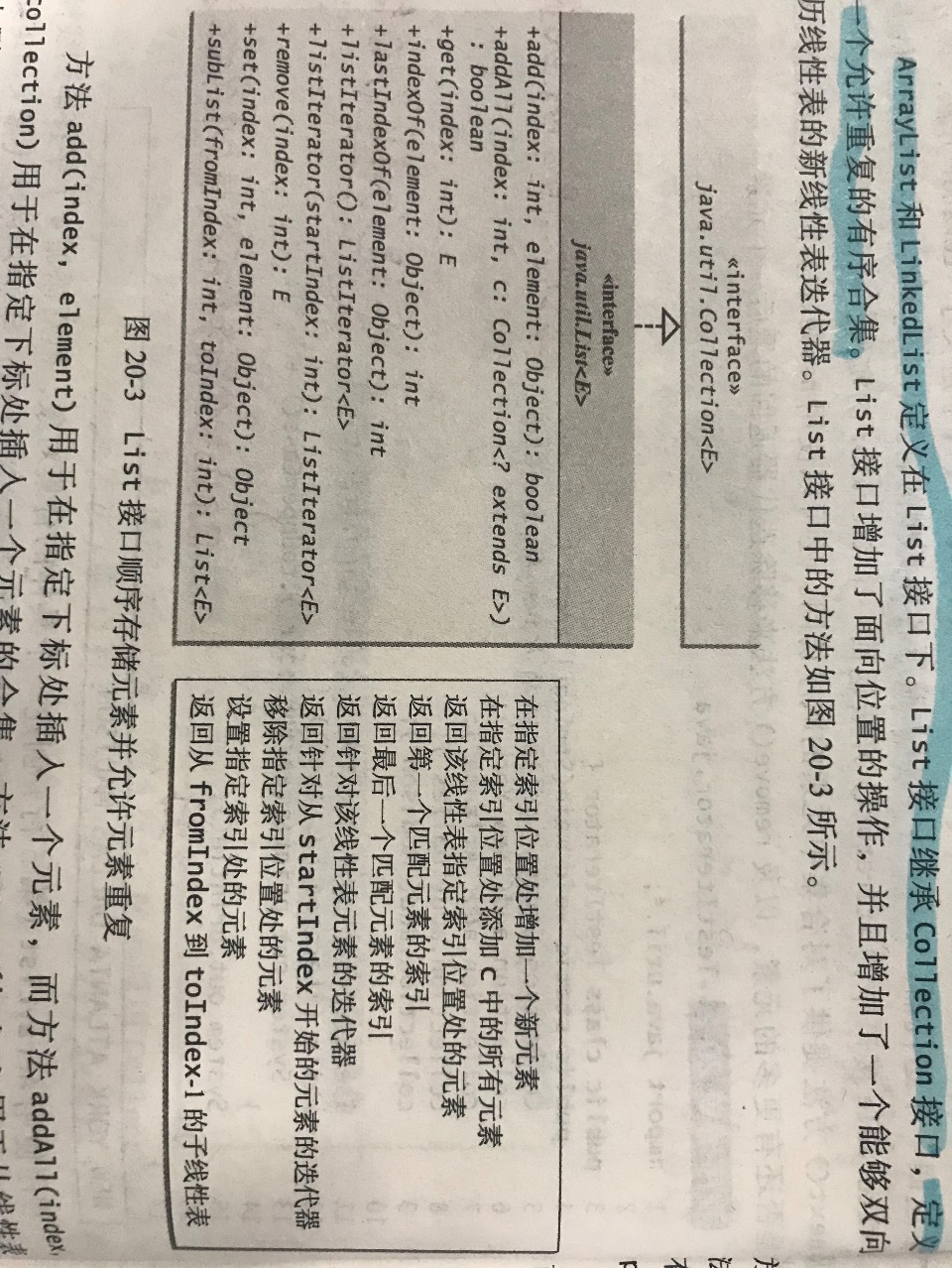
技巧：可以使用foreach循环来简化Interator和while的代码，而不是用迭代器。

## 20.4线性表

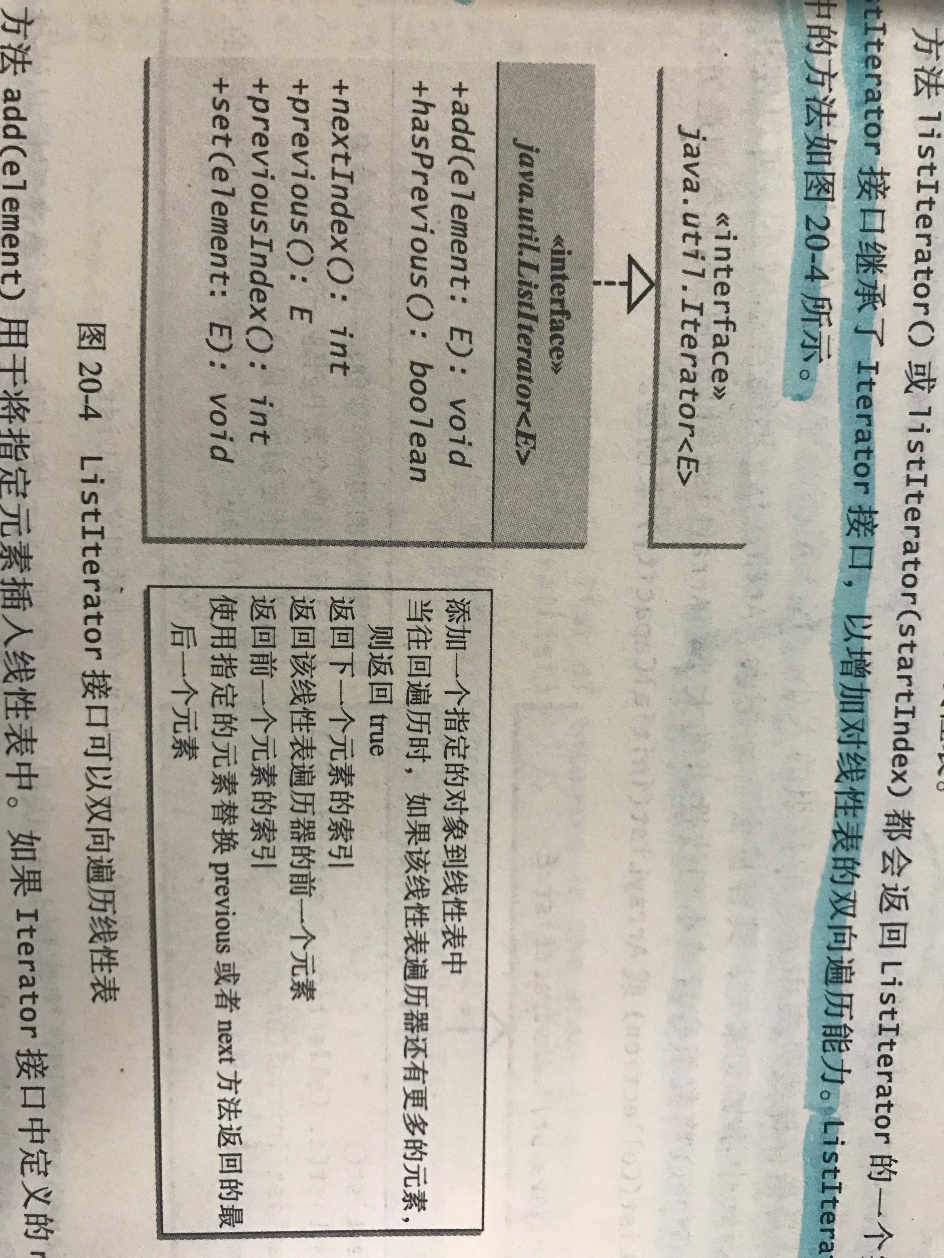
1.要点提示：List接口继承自Collection接口，定义了一个用于顺序存储元素的合集。可以使用它的两个具体类ArrayList或者LinkedList来创建一个线性表（list）。

### 20.4.1 List接口中的通用方法

1.ArrayList和LinkedList定义在List接口下。List接口继承Collection接口，定义了一个允许重复的有序合集。List接口中的方法如下：



2.ListIterator接口继承了Iterator接口，以增加对线性表的遍历能力。ListIterator接口中的方法如下图：

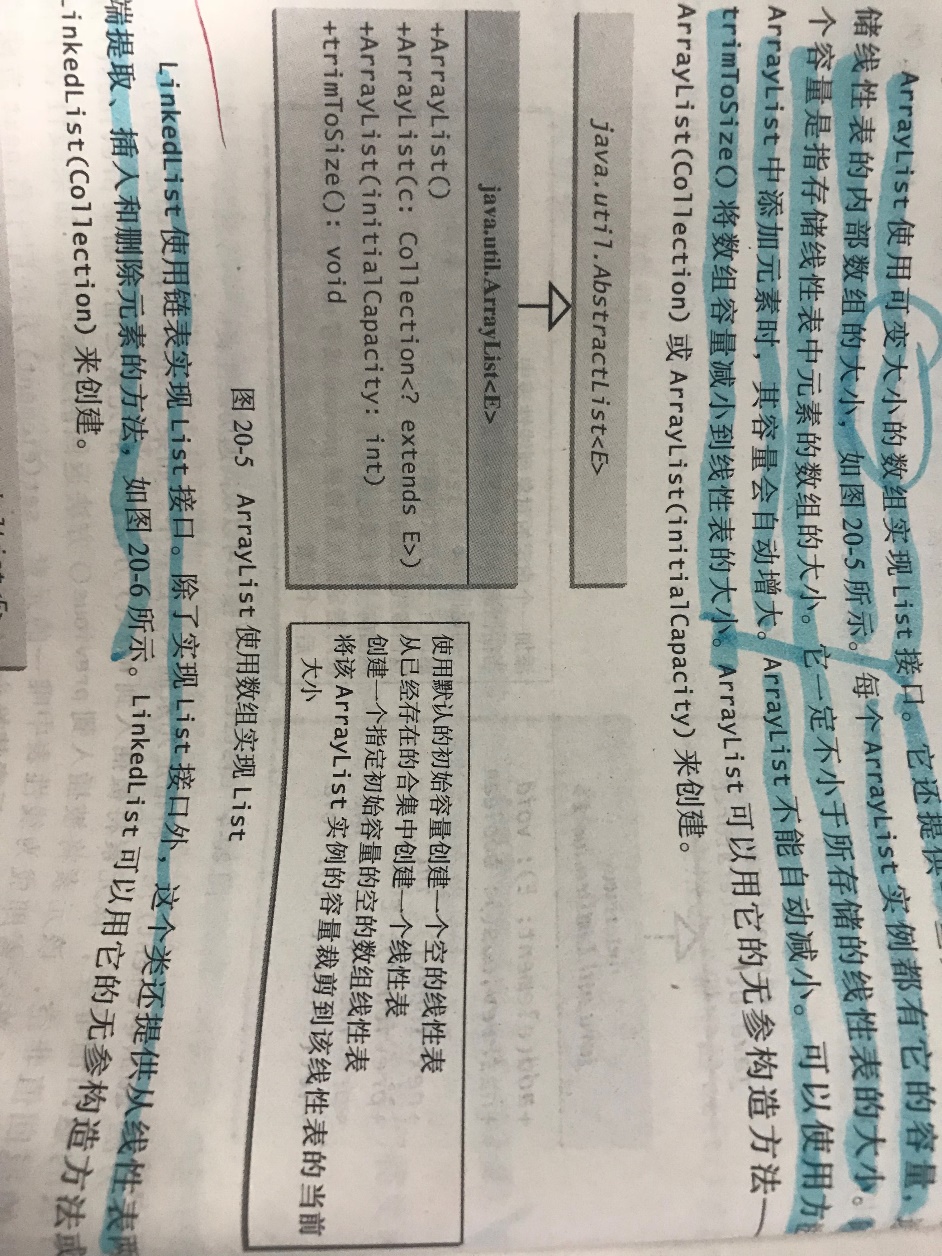


### 20.4.2数组线性表类ArrayList和链表类LinkedList

1.数组线性表类ArrayList和链表类LinkedList是实现List接口的两个具体类。ArrayList用数组存储元素，这个数组是动态创建的。如果元素个数超过了数组的容量，就创建一个更大的新数组，并将当前数组中的所有元素都复制到新数组中。LinkedList在一个链表中存储元素。

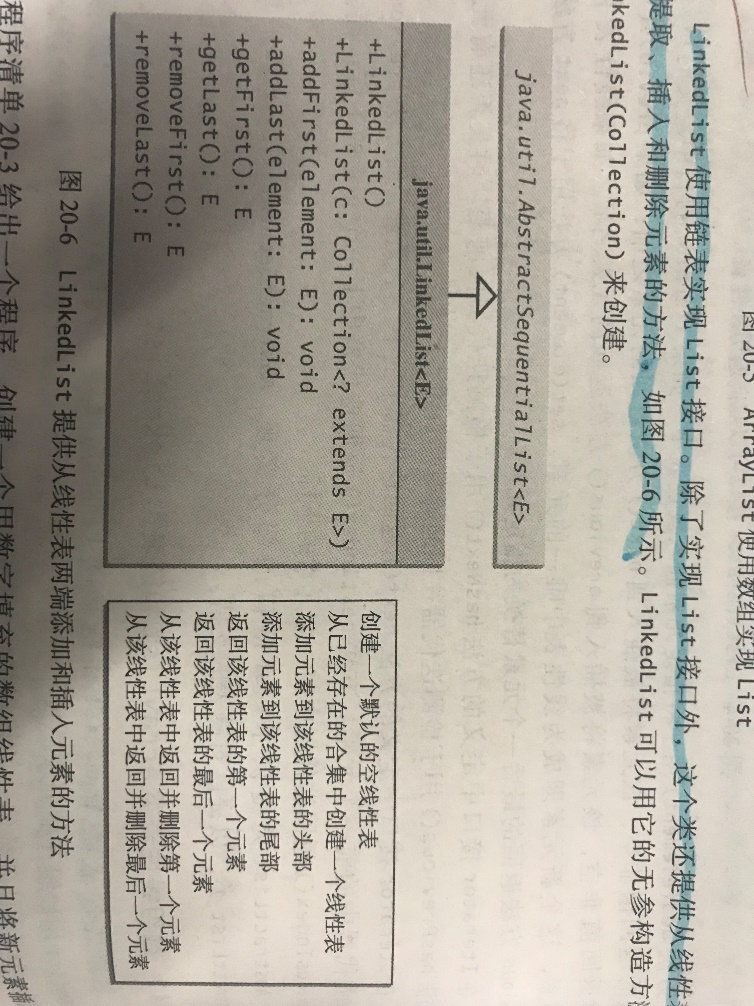
2.要选用这两种中的哪一个依赖于特定的需求。如果需要通过下标随机访问元素，而不会在线性表起始位置插入或删除元素，那么ArrrayList提供了最高效率的集合。但是，如果应用程序需要在线性表的起始位置插入或删除元素，就应该选择LinkedList类。

3.ArrayList使用可变大小的数组实现List接口。它还提供一些方法，用于管理存储线性表内部数组的大小，如图：

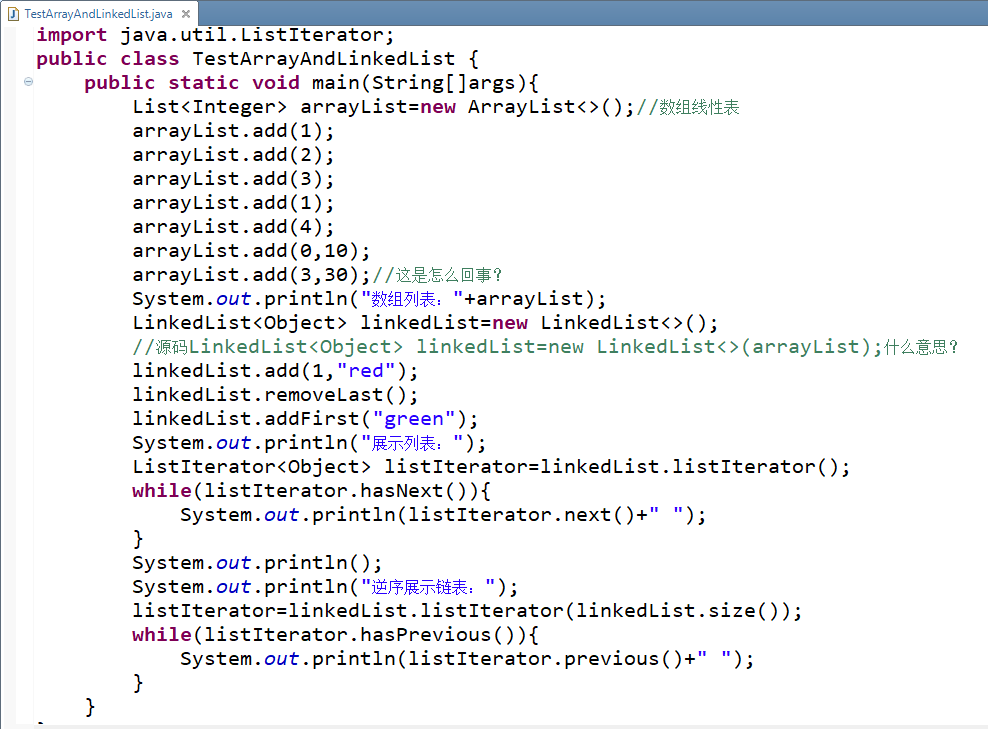


每个ArrayList实例都有它的容量，这个容量是指存储线性表中元素的数组的大小。它一定不小于所存储的线性表的大小。向ArrayList中添加元素时，其容量会自动增大。ArrayList不能自动减小。可以使用方法trimToSize()将数组容量减到线性表的大小。

4.LinkedList使用链表实现List接口。除了实现List接口之外，这个类还提供从线性表两端提取、插入和删除元素的方法，如图：



5.以下程序创建一个数字填充的数组线性表，并且将新元素插入到线性表的指定位置。程序还从数组线性表创建了一个链表，并且向该链表中插入和删除元素。最后，这个程序分别向前、向后遍历该链表。



6.ArrayList和LinkedList比较？

小结：获取元素用ArrayList，插入/删除元素用LinkedList。

ArrayList和LinkedList的操作相似，它们主要的不同体现在内部实现上，内部实现会影响到它们的性能。

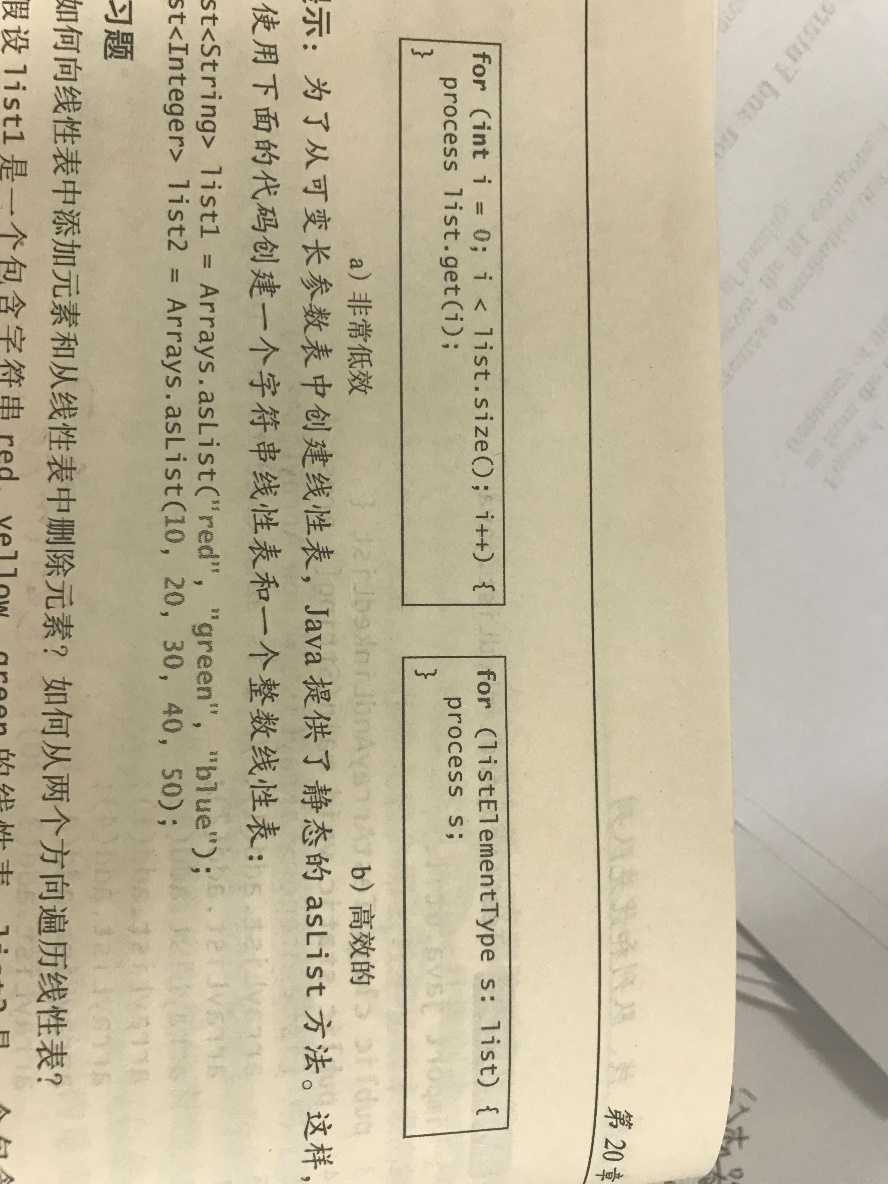
①ArrayList获取元素的效率比较高；

②若在线性表的起始位置插入和删除元素，那么LinkedList的效率会高一些；

③两种线性表在中间或末尾位置上插入和删除元素方面具有相同的性能。

7.链表：

链表可以使用get(i)方法，但是这是一个耗时的操作。不要使用它来遍历线性表中的所有元素，如a所示。应该使用一个迭代器，如b所示。注意foreach循环隐式的使用了迭代器。



## 20.5Comparator接口

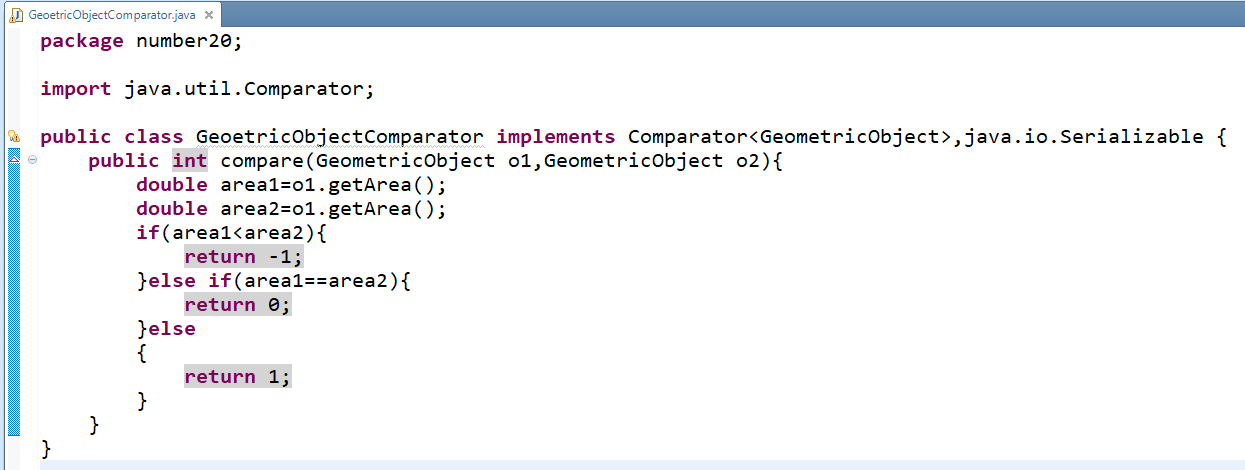
1.要点提示：Comparator可以用来比较没有实现Comparable类的对象。

Java API的一些类，比如String、Date、Calendar、BigInteger以及所有基本类型的数字包装类都实现了Comparable接口。Comparable接口定义了compareTo方法，用于比较实现了Comparable接口的同一类的两个元素。

2.另一种情况？如果元素的类没有实现Comparable接口又将如何呢？可以定义一个比较器来比较不同类的元素。要做到这一点，需要创建一个实现Java.util.Comparator<T>接口的类并重写它的compare方法。

Public int compare (T element1,T element2)

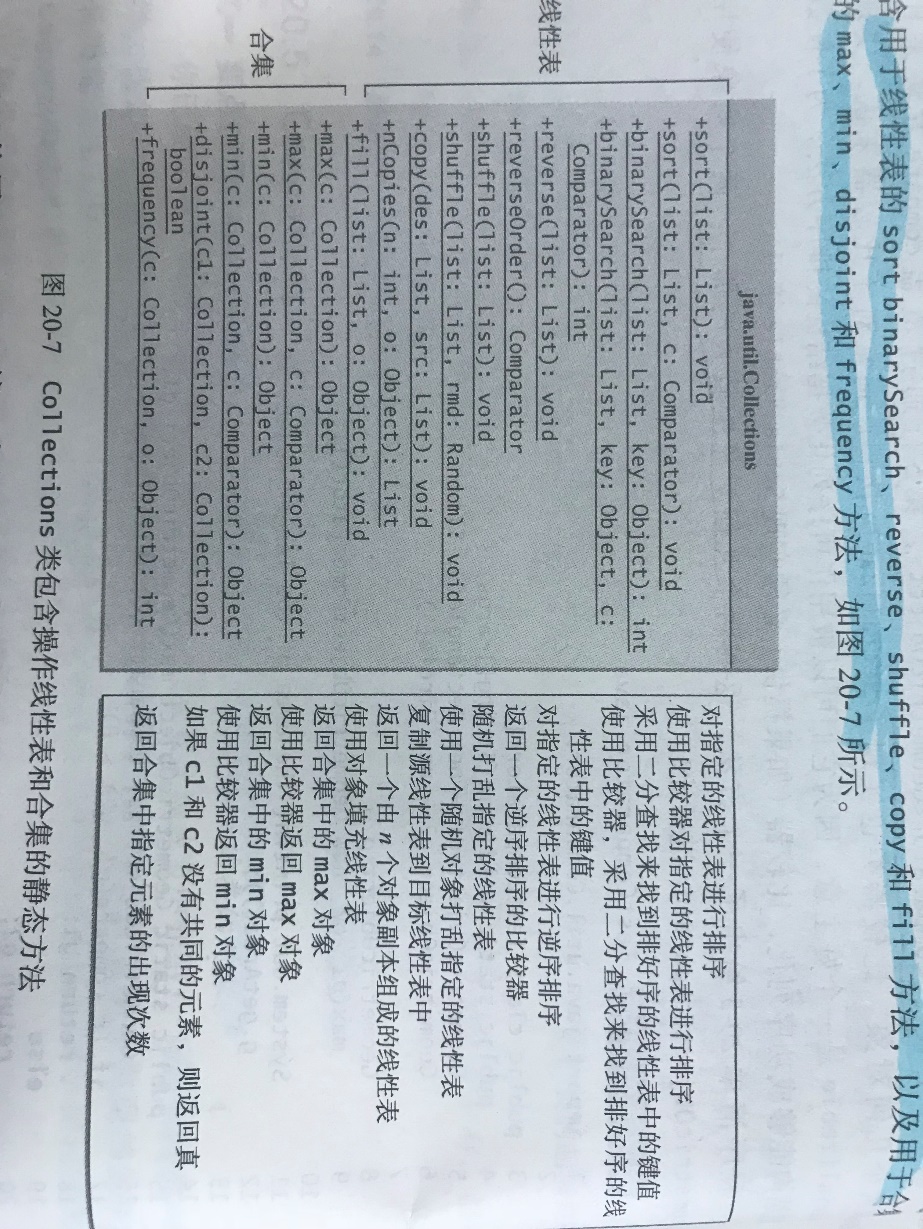
如果element1小于element2，就返回一个负值；如果element1大于element2，就返回一个正值；若两者相等，就返回0.



以下程序给出一个方法，返回两个几何对象中较大的那个，两个对象使用GeometricObjectComparator进行比较：P31程序没写对，拒绝截图。

## 20.6线性表和合集的静态方法

1.要点提示：Collections类包含了执行合集和线性表中通用操作的静态方法。Collections类包含用于线性表的sort binarySearch、reverse、shuffle、copy和fill方法，以及用于合集的max、min、disjoint和frequency方法，如图：



可以使用Comparable接口中的compareTo方法，线性表中可比较的元素以自然顺序排序。也可以指定比较器来对元素排序。例如，下面的代码就是对线性表中的字符串排序：

List<String> list=Arrays.asList(“red”,”green”,”blue”);

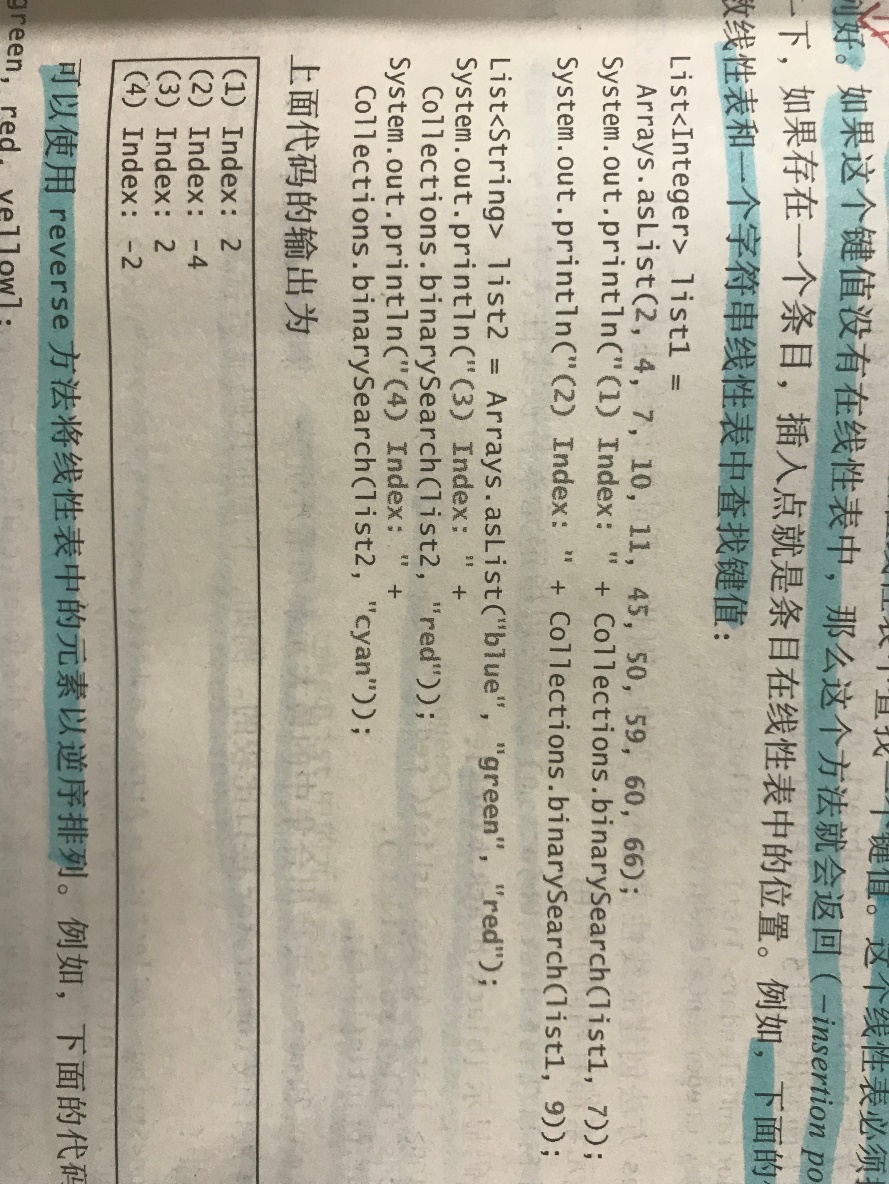
Collections.sort(list);

System.out.println(list);

输出为[blue,green,red]

上面的代码以升序对线性表排序。要以降序排列，可以简单的使用Collections.reverseOrder()方法返回一个Comparator对象，该方法以逆序排列元素。

2.使用binarySearch方法可以在线性表中查找一个键值。这个线性表必须提前以升序排列好。如果这个键值没有在线性表中，那么这个方法就会返回（-insertion point+1）。例如，下面的代码在一个整数线性表和一个字符串线性表中查找键值：



3.可以使用reverse方法将线性表中的元素以逆序排列。

Collections.reverse(list);

4.可以使用shuffle(list)方法对线性表中的元素进行随机重新排序

Collections.shuffle(list);

5.可以使用copy（det，src）方法将源线性表中所有的元素以同样的下标复制到目标线性表中。目标线性表必须跟源线性表等长。如果线性表的长度大于目标线性表，那么，目标线性表中剩余元素不会受到影响。例如，下面将list2复制到list1中：

List<String>list1=Arrays.asList(“yellow”,”red”,”green”,”blue”);

List<String>list2=Arrays.asList(“white”,”black”);

6.可以使用fill(List list,Object o)方法来指定元素替换线性表中所有元素。例如，下面的代码显示[black,black,black]:

List<String>list=Arrays.asList(“red”,”green”,”blue”);

Collections.fill(list,”black”);

System.out.println(list);

7.可以使用max和min方法找出合集中的最大最小元素。集合中的元素必须是可使用Comparable接口或Comparator接口比较的的。例如，下面的代码显示和集中的最大和最小字符串：

Collection<String>collection=Arrays.asList(“red”,”green”,”blue”);

System.out.println(Collection.max(collection));

20.7示例学习：弹球