**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。**

**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合集，简称为合集(collection)。**

**2.另一种是为了存储键/值对，称为映射表(map)。**

**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。**

**提供一个部分实现接口的抽象类对用户编写代码提供了方便。用户可以简单地定义一个具体类继承自抽象类，而无须实现接口中的所有方法。为了方便，提供了入AbstractCollection这样的抽象类。因为这个原因，这些抽象类被称为遍历抽象类(convenience abstract class).**

**Collection接口中的有些方法是不能再具体子类中实现的。在这样的情况下，这些方法会抛出异常java.lang.UnsupportedOperationException，它是RuntimeException异常类中的一个子类。可以这样使用，如果一个方法在子类中没有意义，可以这样实现：**

**public void someMethod(){**

**throw new UnsupportedOperationException("Method not supported");**

**}**

**迭代器**

**每种合集都是可迭代的(Iterable)。可以获得集合的Iterator对象来遍历合集中的所有元素**

**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再数据结构的细节的情况下，来遍历一个数据结构。**

**Collectioin接口继承自Iterable接口。Iterable接口中定义了iterator方法，该方法会返回一个迭代器。例如**

**Collection<String> collection=new ArrayList<>();**

**collection.add("New York");**

**collection.add("Dallas");**

**Iterator<String> iterator=collection.iterator();**

**while(iteraotr.hasNext()){**

**System.out.print(iterator.next()+" ");**

**}**

**线性表**

**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(linked list)中存储元素。**

**选择哪一个依赖于特定需求。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ArrayList提供了最高效率的合集。但是，如果应用程序要在线性表的起始位置上插入或删除元素，就应该选择LinkedList类。**

**线性表的大小是可以动态增大或减小的。然而数组一旦被创建，它的大小就是固定的。**

**ArrayList不能自动减小，可以使用方法trimToSize()将数组容量减小到线性表的大小。**

**ArrayList可以用它的无惨构造方法ArrayList(Collection)或ArrayList(initialCapacity)来创建。**

**线性表可以存储相同的元素。**

**ArrayList获取元素的效率比较高；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。两种线性表在中间或者末尾位置上插入和删除元素方面具有同样的性能。**

**链表可以使用get(i)方法，但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，应该使用一个迭代器。foreach循环隐式地使用了迭代器。**

**for (listElementType s:List){**

**process s;**

**}**

**为了从可变长参数列表中创建线性表，Java提供了静态的asList方法。**

**List<String> list1=Arrays.asList("red","green","ble"); // 创建字符串线性表**

**List<Integer> list2=Arrays.asList(10,20,30); // 创建一个整数线性表**

**Comparator接口**

**Comparable用于比较实现Comparable的类的对象；Comparator用于比较没有实现Comparable的类的对象。**

**使用Comparable接口来比较元素称为使用自然顺序进行比较，使用Comparator接口来比较元素称为使用比较器来进行比较**

**GeometricObject g1=new Rectangle(5,5);**

**GeometricObject g2=new Circle(5);**

**GeometricObject g=max(g1,g2,new GeometricObjectComparator()); //这里传入一个比较器GeometricObjectCompar**

**GeometricObject max (GeometricObject g1, GeometricObject g2, //方法 Comparator<GeometricObject> c) {**

**if( c.compare(g1,g2) >0)**

**return g1;**

**else**

**return g2;**

**}**

**class GeometricObjectComparator //比较器 实现comparator接口，覆盖compare方法来比较两个几何对象，实现了Serializable接口**

**implements Comparator <GeometricObject> ,java.io.Serializable {**

**public int compare(GeometricObject o1, GeometricObject o2 ){**

**double area1=o1.getArea();**

**double area2=o2.getArea();**

**if(area1<area2)**

**return -1;**

**else if(area1==area2)**

**return 0;**

**else**

**return 1;**

**}**

**}**

**通常对于比较器来说实现Serializable是一个好主意，因为它们可以被用作可序列化数据结构的排序方法，为了使数据结构能够成功序列化，比较器（如果提供）必须实现Serializable接口。**

**可以使用Comparable接口中的compareTo方法，对线性表中的可比较的元素以自然顺序排列。也可以制定比较器来对元素排序。例如**

**List <String> list= ArrayList.asList("red","green","blue");**

**Collections.sort( list ); // Collection类通用操作的静态方法，此为升序**

**System.out.println(list);**

**Collections.reverseOrder()方法返回一个Comparator对象，该方法以逆序排列元素**

**Collections.sort(list, Collections.reverserOrder()); // 此为降序**

**向量类和栈类**

**vector类是AbstractList的子类，Stack是Vector的子类**

**除了包含用于访问和修改向量的同步方法之外，Vector类与ArrayList类是一样的。同步方法用于防止两个或多个线程同时访问和修改某个向量引起数据损坏。对于许多不需要同步的应用程序来说，使用ArrayList比使用Vector效率更高。**

**队列和优先队列**

**在优先队列（priority queue）中，元素被赋予优先级。当访问元素时，拥有最高优先级的元素首先被删除。**

**Queue接口继承自java.util.Collection，加入了插入、提取和检验等操作。**

**双端队列Deque和链表LinkedList：**

**LinkedList类实现了Deque接口，Deque有继承自Queue接口，因此可以使用LInkedList创建一个队列。LinkedLIst很适合用于进行队列操，因为它可以有效地在线性表的两端插入和移除元素。**

**使用队列存储字符串的例子**

**public class TestQueue {**

**public static void main( String[] args ) {**

**java.util.Queue <String> queue=new java.util.LinkedLIst<>();**

**queue.offer("Oklahoma");**

**queue.offer("Indiana");**

**while (queue.size()>0)**

**System.out.print (queue.remove()+" ");**

**}**

**}**

**PriorityQueue 类实现了一个优先队列，默认情况下，优先队列使用Comparable以元素的自然顺序进行排序。拥有最小数值的元素被赋予最高优先级，因此最先从队列中删除。如果几个元素具有相同的最高优先级，则任意选择一个。也可以使用构造方法PriorityQueue( initialAcpacitty, comparator)中的Comparator来指定一个顺序。**

**//使用Comparable自然顺序排序**

**PriorityQueue<String> queue1=new PriorityQueue<>();**

**//使用从Collections.reverseOrder()中获得的比较器创建优先队列，该方法以逆序排序**

**PriorityQueue<String> queue2=new PriorityQueue(4,Collections.reverseOrder());**

**小结**

**1.java合集框架支持集合、线性表、队列和映射表，他们分别定义在接口Set、List、Queue和Map中。**

**2.线性表用于存储一个有序的元素合集。**

**3.除去PriorityQueue，java合集框架中的所有实例类都实现了CLoneable和Serializable接口，所以，它们的实例都是可以可克隆和可序列化的。**

**4.若要在合集中存储重复的元素，就需要使用线性表。线性表不仅可以存储重复的元素，而且允许用户指定存储的位置。用户可以通过下标来访问线性表中的元素。**

**5.Java合集框架支持两种类型的线性表：数组线性表ArrayList和链表LinkedList。ArrayList是实现LIst接口的可变大小的数组。ArrayList中的所有方法都是在List接口中定义的。LinkedList是实现LIst接口的一个链表。除了实现了LIst接口，该类还提供了可从线性表两端提取、插入以及删除元素的方法。**

**6.Comparator可以用于比较没有实现Comparable接口的类的对象。**

**7.Vector类继承了AbstractList类。从java2开始，Vector类和ArrayList是一样的，所不同的是它所包含的访问和修改向量的方法是同步的。Stack类是继承了Vector类，并且提供了几种对栈进行操作的方法。**

**8.Queue接口表示队列。PrioriQueue类为优先队列实现了Queue接口。**