**集合（set）是一个用于存储和处理无重复元素的高效数据结构。映射表（map）类似于目录，提供了使用键值快速查询和获取值的功能。**

**集合**

**可以使用集合的三个具体类HashSet、LinkedHashSet、TreeSet来创建集合。**

**Set接口扩展了Collection接口，它没有引入新的方法或敞亮，只是规定Set的实例不包含重复的元素。即在一个集合中，不存在元素e1和e2，使得e1.equals( e2 )的返回值为true。**

**AbstractSet类继承AbstractCollection类并部分实现Set接口。AbstractSet类提供equals方法和hashCode方法的具体实现。**

**一个集合的散列码是这个集合中所有元素散列码的和。**

**Set接口的三个具体类是：散列类HashSet、链式散列集LinkedHashSet和树形集TreeSet**

**HashSet**

**HashSet类是一个实现lSet接口的具体类，可以使用它的无参构造方法来创建空的散列集，也可以由一个现有的合集创建散列集。默认情况下，初始容量为16而负载系数是0.75.负载系数的值在0.0-1.0之间。**

**负载系数（load factor）测量该集合允许多满。当元素个数超过了容量与负载系数的乘积，容量就会翻倍。例如当尺寸达到16\*0.75=12时，容量将会翻到32.**

**比较高的负载系数会降低空间开销，但是会增加查找时间。通常情况下，默认的负载系数是0.75，它是在时间开销和空间开销上一个很好的权衡。**

**如果两个对象相等，那么这两个对象的散列码必须一样。两个不相等的对象可能会有相同的散列码。**

**Integer类中的hashCode方法返回它的int值，Character类中的hashCode方法返回这个字符的统一码，String类中的hashCode方法返回s0\*31^(n-1)+s1\*31^(n-2)+...+s(n-1),其中si是s.charAt(i)。**

**字符串不按照它们插入集合时的顺序存储，因为散列集中的元素是没有特定的顺序的。要强加给它们一个顺序，就需要使用LInkedHashSet类。**

**Collection接口继承Iterable接口，因此集合中的元素时可遍历的。使用foreach循环来遍历集合中的所有元素。例如**

**import java.util.\*;**

**public class TestHashSet {**

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

**// Create a hash set**

**Set<String> set = new HashSet<>();**

**// Add strings to the set**

**set.add("London");**

**set.add("Paris");**

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

**set.add("San Francisco");**

**set.add("Beijing");**

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

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

**// Display the elements in the hash set**

**for (String s: set) {**

**System.out.print(s.toUpperCase() + " ");**

**}**

**// Process the elements using a forEach method**

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

**set.forEach(e -> System.out.print(e.toLowerCase() + " "));**

**}**

**}**

**LInkedHashSet**

**LinkedHashSet用一个链表实现来扩展HashSet类，它支持对集合内的元素排序。**

**例如**

**public class TestLinkedHashSet {**

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

**// Create a hash set**

**Set<String> set = new LinkedHashSet<>();**

**// Add strings to the set**

**set.add("London");**

**set.add("Paris");**

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

**set.add("San Francisco");**

**set.add("Beijing");**

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

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

**// Display the elements in the hash set**

**for (String element: set)**

**System.out.print(element.toLowerCase() + " ");**

**}**

**}**

**如果不需要维护元素被插入的顺序，就应该使用HashSet，它会比LinkedHashSet更高效。**

**TreeSet**

**SortedSet是Set的一个子接口，它可以确保集合中的元素时有序的。另外，还提供方法first()和last()以返回集合中第一个元素和最后一个元素。**

**NavigableSet扩展了SortedSet，并提供导航方法lower(e),floor(e)..**

**TreeSet实现了SortedSet接口。只要对象是可以相互比较的，就可以将它们添加到一个树形集中。**

**例如**

**public class TestTreeSet {**

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

**// Create a hash set**

**Set<String> set = new HashSet<>();**

**// Add strings to the set**

**set.add("London");**

**set.add("Paris");**

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

**set.add("San Francisco");**

**set.add("Beijing");**

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

**TreeSet<String> treeSet = new TreeSet<>(set);**

**System.out.println("Sorted tree set: " + treeSet);**

**// Use the methods in SortedSet interface**

**System.out.println("first(): " + treeSet.first());**

**System.out.println("last(): " + treeSet.last());**

**System.out.println("headSet(\"New York\"): " +**

**treeSet.headSet("New York"));**

**System.out.println("tailSet(\"New York\"): " +**

**treeSet.tailSet("New York"));**

**// Use the methods in NavigableSet interface**

**System.out.println("lower(\"P\"): " + treeSet.lower("P"));**

**System.out.println("higher(\"P\"): " + treeSet.higher("P"));**

**System.out.println("floor(\"P\"): " + treeSet.floor("P"));**

**System.out.println("ceiling(\"P\"): " + treeSet.ceiling("P"));**

**System.out.println("pollFirst(): " + treeSet.pollFirst());**

**System.out.println("pollLast(): " + treeSet.pollLast());**

**System.out.println("New tree set: " + treeSet);**

**}**

**}**

**java合集框架中的所有具体类都至少有两个构造方法：一个是建立空合集的无参构造 方法，另一个是用某个合集来创建实例的构造方法。这样，TreeSet类中就含有从合集c创建TreeSet对象的构造方法TreeSet（Collection c）。在这个例子中，new TreeSet<>(set)方法从合集set创建了TreeSet的一个实例。**

**当更新一个集合时，如果不需要保持元素的排序关系，就应该使用散列集，以你为在散 列集中插入和删除元素所花的时间比较少。当需要一个排好序的合集时，可以从这个散列集创建一个树形集。**

**如果使用无参构造方法创建一个TreeSet，则会假定元素的类实现了Comparable接口，并使用compareTo方法来比较集合中的元素。要使用comparator，则必须用构造方法TreeSet（Comparator comparator），使用比较器中的compare方法来创建一个排好序的集合。例如**

**public class TestTreeSetWithComparator {**

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

**// Create a tree set for geometric objects using a comparator**

**Set<GeometricObject> set =**

**new TreeSet<>(new GeometricObjectComparator());**

**set.add(new Rectangle(4, 5));**

**set.add(new Circle(40));**

**set.add(new Circle(40));**

**set.add(new Rectangle(4, 1));**

**// Display geometric objects in the tree set**

**System.out.println("A sorted set of geometric objects");**

**for (GeometricObject element: set)**

**System.out.println("area = " + element.getArea());**

**}**

**}**

**GeometricObjectComparator类在别的地方定义。程序创建了一个集合对象的树形集，并使用GeometricObjectComparator来比较集合中的元素。**

**集合内不允许有重复的元素**

**比较集合和线性表的性能**

**在无重复元素进行排序方面，集合比线性表更加高效。线性表在通过索引来访问元素方面非常有用。**

**线性表中的元素可以通过索引来访问。而集合不支持索引，因为集合中的元素时无序的。要遍历集合中的所有元素， 使用foreach循环。**

**在测试一个元素是否在集合或者线性表的方面，集合比线性表更加高效。**

**例如，测试一个单词是否在关键字集合中。**

**public class CountKeywords {**

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

**Scanner input = new Scanner(System.in);**

**System.out.print("Enter a Java source file: ");**

**String filename = input.nextLine();**

**File file = new File(filename);**

**if (file.exists()) {**

**System.out.println("The number of keywords in " + filename**

**+ " is " + countKeywords(file));**

**}**

**else {**

**System.out.println("File " + filename + " does not exist");**

**}**

**}**

**public static int countKeywords(File file) throws Exception {**

**// Array of all Java keywords + true, false and null**

**String[] keywordString = {"abstract", "assert", "boolean",**

**"break", "byte", "case", "catch", "char", "class", "const",**

**"continue", "default", "do", "double", "else", "enum",**

**"extends", "for", "final", "finally", "float", "goto",**

**"if", "implements", "import", "instanceof", "int",**

**"interface", "long", "native", "new", "package", "private",**

**"protected", "public", "return", "short", "static",**

**"strictfp", "super", "switch", "synchronized", "this",**

**"throw", "throws", "transient", "try", "void", "volatile",**

**"while", "true", "false", "null"};**

**Set<String> keywordSet =**

**new HashSet<>(Arrays.asList(keywordString));**

**int count = 0;**

**Scanner input = new Scanner(file);**

**while (input.hasNext()) {**

**String word = input.next();**

**if (keywordSet.contains(word))**

**count++;**

**}**

**return count;**

**}**

**}**

**映射表**

**映射表（map）是一种依照键/值对存储元素的容器。它提供了通过键快速获取、删除和更新键/值对的功能。映射表将值和键一起保存。键很像下标。在List中，下标是整数；而在Map中，键可以是任意类型的对象。**

**映射表中不能有重复的键，每个键都对应一个值。一个键和它的对应值构成一个条目并保存在映射表中。**

**Map接口提供了查询、更新和获取合集的值和合集的键的方法。**

**更新方法（update method）包括clear、put、putAll、remove。**

**方法clear()从映射表中删除所有的条目。方法put(K key,V value)为映射表中指定的键和值添加条目。如果这个映射表原来就包含该键的一个条目，则原来的值将被新的值所替代，并且返回与这个键相关联的原来的值。方法putAll(Map m)将m中的所有条目添加到这个映射表中。方法remove(Object key)将指定键对应的条目从映射表中删除。**

**查询方法（query method）包括containsKey、containsValue、isEmpty和size。方法containsKey(Object key)检测映射表中是否包含指定键的条目。方法containsValue(Object value)检测图中是否包含指定值的条目。方法isEmpty()检测映射表中是否包含条目。方法size()返回映射表中条目的个数。**

**可以使用方法keySet()来获得一个包含映射表中键的集合，也可以是使用方法values()获得一个包含映射表中值的合集。方法entrySet()返回一个所有条目的集合。这些条目是Map.Entry<K,V>接口的实例，这里Entry是Map接口的一个内部接口。**

**AbstractMap类是一个遍历抽象类，它实现了Map接口中除了entrySet()方法之外的所有方法。**

**HashMap、LinkedHashMap和TreeMap类是Map接口的三个具体实现（concrete implementation)。**

**对于定位一个值、插入一个条目以及删除一个条目而言，HashMap类是高效的。**

**HashMap类中的条目是没有顺序的，但是在LinkedHashMap中，元素既可以按照他们插入映射表的顺序排序（称为插入顺序（insertion order）），也可以按它们被最后一次访问时的顺序，从最早到最晚（称为访问顺序（access order））排序。**

**无参构造方法是以插入顺序来创建LinkedHashMap对象的。要按访问顺序创建LinkedHashMap对象，应该使用构造方法LinkedHashMap( initialCapacity,loacFactor,true)。**

**TreeMap类在遍历排好顺序的键时是很高效的。键可以使用Comparable接口或Comparator接口来排序。要使用比较器，必须使用构造方法TreeMap(Comparator comparator)来创建一个有序映射表，这样，该映射表中的条目就能使用比较器中的compare方法按键进行排序。**

**SortedMap是Map的一个子接口，使用它可确保映射表中的条目是排好序的。还提供firstKey()和lastKey()来返回映射表中的第一个和最后一个键，还有headMap(toKey)和tailMap(fromKey)。**

**NavigableMap继承了SortedMap，以提供导航方法lowerKey(key)、floorKey(key)、ceilingKey(key)和higherKey(key)来分别返回小于、小于或等于、大于或等于、大于某个给定键的键，如果没有这样的键，则都返回null。方法pollFirstEntry()和pollLastEntry()分别删除并返回树映射表中的第一个和最后一个条目。**

**例子创建一个散列映射表（hash map）、一个链式散列映射表（linked hash map）和一个树形映射表（tree map），以建立学生与年龄之间的映射关系。**

**public class TestMap {**

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

**// Create a HashMap**

**Map<String, Integer> hashMap = new HashMap<>();**

**hashMap.put("Smith", 30);**

**hashMap.put("Anderson", 31);**

**hashMap.put("Lewis", 29);**

**hashMap.put("Cook", 29);**

**System.out.println("Display entries in HashMap");**

**System.out.println(hashMap + "\n");**

**// Create a TreeMap from the preceding HashMap**

**Map<String, Integer> treeMap = new TreeMap<>(hashMap);**

**System.out.println("Display entries in ascending order of key");**

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

**// Create a LinkedHashMap**

**Map<String, Integer> linkedHashMap =**

**new LinkedHashMap<>(16, 0.75f, true);**

**linkedHashMap.put("Smith", 30);**

**linkedHashMap.put("Anderson", 31);**

**linkedHashMap.put("Lewis", 29);**

**linkedHashMap.put("Cook", 29);**

**// Display the age for Lewis**

**System.out.println("\nThe age for " + "Lewis is " +**

**linkedHashMap.get("Lewis"));**

**System.out.println("Display entries in LinkedHashMap");**

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

**// Display each entry with name and age**

**System.out.print("\nNames and ages are ");**

**treeMap.forEach(**

**(name, age) -> System.out.print(name + ": " + age + " "));**

**}**

**}**

**HashMap中条目的顺序是随机的，而TreeMap中的条目是按键的升序排列的，LinkedHashMap中的条目则是按元素最后一次被访问的时间从早到晚排序的。**

**实现Map接口的所有具体类至少有两种构造方法：一种是无参构造方法，它可用来创建一个空映射表，而另一种构造方法是从Map的一个实例来创建映射表。所以，语句new TreeMap <String,Integer>(hashMap)就是从一个散列映射表来创建一个树形映射表。**

**如果更新映射表时不需要保持映射表中元素的顺序，就使用HashMap；如果需要保持映射表中元素的插入顺序或访问顺序，就是用LInkedHashmap；如果需要使映射表按照键排序，就使用TreeMap。**

**单词的出现次数，例子**

**public class CountOccurrenceOfWords {**

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

**// Set text in a string**

**String text = "Good morning. Have a good class. " +**

**"Have a good visit. Have fun!";**

**// Create a TreeMap to hold words as key and count as value**

**Map<String, Integer> map = new TreeMap<>();**

**String[] words = text.split("[\\s+\\p{P}]");**

**for (int i = 0; i < words.length; i++) {**

**String key = words[i].toLowerCase();**

**if (key.length() > 0) {**

**if (!map.containsKey(key)) {**

**map.put(key, 1);**

**}**

**else {**

**int value = map.get(key);**

**value++;**

**map.put(key, value);**

**}**

**}**

**}**

**// Display key and value for each entry**

**map.forEach((k, v) -> System.out.println(k + "\t" + v));**

**}**

**}**

**单元素与不可变的合集和映射表**

**可以使用Collectionis类中的静态方法来创建单元素的集合、线性表和映射表，以及不可变集合、线性表和映射表。**

**小结**

**1.集合存储的是不重复的元素。若要在合集中存储重复的元素，需要使用线性表。**

**2.映射表中存储的是键/值对。它提供使用键快速查询一个值。**

**3.java合集框架支持三种类型的集合：散列集HashSet、链式散列集LInkedHashSet和树形集TreeSet。HashSet以一个不可预知的顺序存储元素；LInkedHashSet以元素被插入的顺序存储元素；TreeSet存储以排好序的元素。HashSet、LInkedHashSet和TreeSet中的所有方法都继承自Collection接口。**

**4.Map接口将键映射到元素上。键类似索引。LIst中，索引为整数。Map中，键可以为任何对象。映射表不能包含相同的键。每个键可以映射最多一个值。Map接口提供了查询、更新一集获取值的合集一集键的集合的方法。**

**5.java合集框架支持三种类型的映射表：散列映射表HashMap、链式散列映射表LinkedHashMap和树形映射表TreeMap。对于定位一个值、插入一个条目和删除一个条目而言，HashMap是很高效的。LInkedHashMap支持映射表中的条目排序。HashMap类中的条目是没有顺序的，但LinkedHashMap中的条目可以按某种顺序来获取，该顺序既可以是它们被插入映射表中的顺序（称为插入顺序），也可以是它们最后一次被访问的时间的顺序，从最早到最晚（称为访问顺序）。对于遍历排好序的键，TreeMap是高效的。键可以使用Comparable接口来排序，也可以使用Comparator接口来排序。**