第 5 章　　容　器　类

BACK

5.1 容器的概念与相互关系

Java容器类的对象实例又称为容器，容器用于保存对象。根据所存储的元素的形式，可将容器按照接口不同划分为两大类：

（1）Collection：一个由一系列元素组成的序列，其中的元素是指向其他对象的引用。

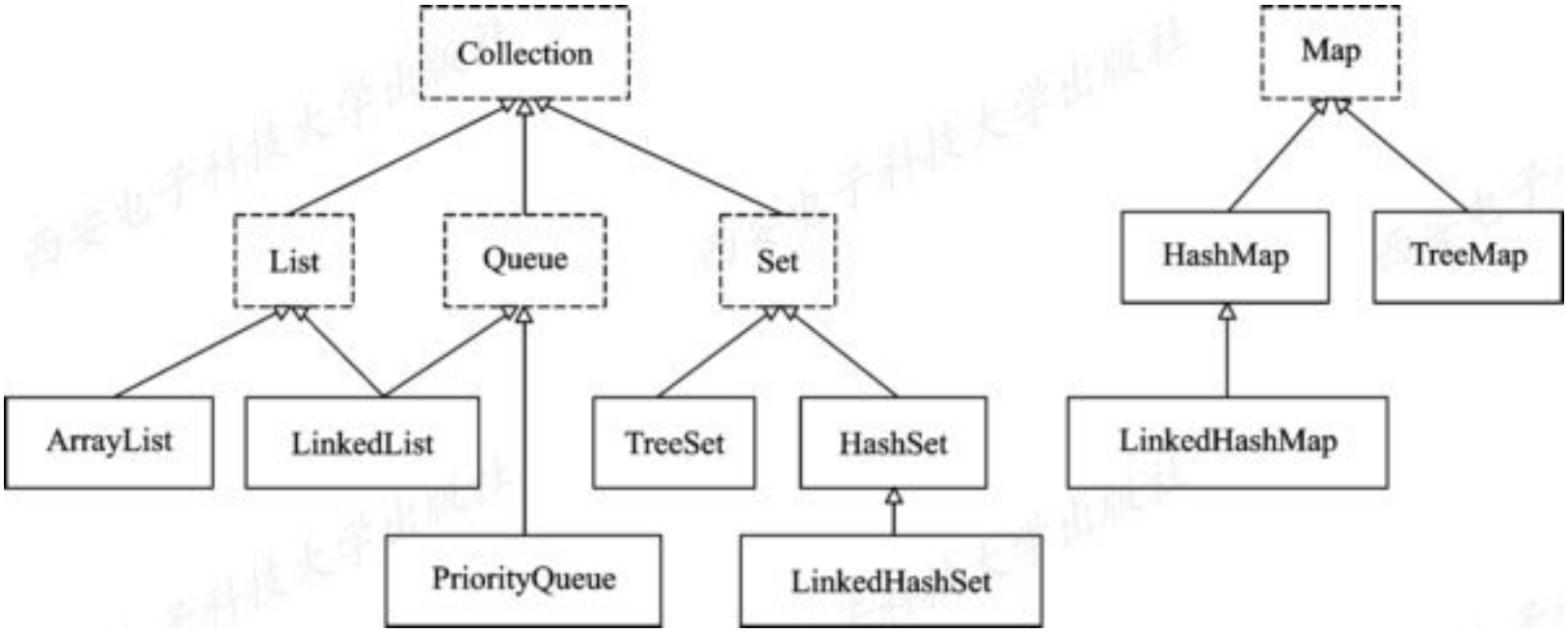

图5-1　容器层次的简要结构

例 5－1　TestCollection.java：Collection 示例。

```
public class TestCollection{
    Collection<String> fill(Collection<String> col){
        col.add("Java");
        col.add("C");
        col.add("Python");
        col.add("Python");
        return col;
    }
    Map<String, Integer> fill(Map<String, Integer> map){
        map.put("Java", 19);
        map.put("C", 42);
        map.put("Python", 100);
        map.put("Python", 101);
        return map;
    }
```

```
    public static void main(String[] args){
        TestCollection tc=new TestCollection();
        System.out.println(tc.fill(new ArrayList<String>()));
        System.out.println(tc.fill(new LinkedList<String>()));
        System.out.println(tc.fill(new HashSet<String>()));
        System.out.println(tc.fill(new TreeSet<String>()));
        System.out.println(tc.fill(new LinkedHashSet<String>()));
        System.out.println(tc.fill(new HashMap<String, Integer>()));
        System.out.println(tc.fill(new TreeMap<String, Integer>()));
        System.out.println(tc.fill(new LinkedHashMap<String, Integer>()));
    }
}
```

```
/*输出结果：
[Java, C, Python, Python]
[Java, C, Python, Python]
[C, Python, Java]
[C, Java, Python]
[Java, C, Python]
{C=42, Python=101, Java=19}
{C=42, Java=19, Python=101}
{Java=19, C=42, Python=101}
*/
```

5.2 Set接口及其实现

Set不接受重复的元素。Java SE中提供3种Set容器的实现：HashSet、TreeSet和LinkedHashSet。HashSet类采用Hash表实现Set接口，它的查询速度最快，但其中的元素没有固定顺序；TreeSet类采用红黑树结构实现了SortedSet接口，能够保证元素处于排序状态；LinkedHashSet类采用Hash表与链表结合的方式实现Set接口，它能够以插入顺序保存元素。

表 5-1　Collection 与 Set 的主要接口方法及其含义

接口方法	含　义
boolean add(E e)	如果给定元素不存在于当前 Set 中，则将其加入当前 Set
boolean addAll(Collection<? extends E> c)	对于一个容器 c 中的所有元素，如果它不存在于当前 Set 中，则将其加入当前 Set。只要添加了任意元素就返回 true，否则返回 false
void clear()	移除当前 Set 中的所有元素

续表

接口方法	含 义
boolean contains(Object o)	如果指定元素在当前 Set 中，则返回 true；否则返回 false
boolean containsAll(Collection<? > c)	对于一个容器 c，如果当前 Set 包含该容器中的所有元素，则返回 true；否则返回 false
boolean isEmpty()	如果当前 Set 不包含任何元素，则返回 true；否则返回 false
boolean remove(Object o)	如果指定元素在当前 Set 中，则移除该元素
boolean removeAll(Collection<? > c)	从当前 Set 中将指定容器中包含的所有元素都移除。只要有移除动作发生就返回 true；否则返回 false
boolean retainAll(Collection<? > c)	只保留当前 Set 中同时也包含于指定容器中的元素。集合发生了改变就返回 true；否则返回 false
int size()	返回当前 Set 中元素的数量
Object[] toArray()	返回一个数组，该数组由当前 Set 中所有元素组成
<T> T[] toArray(T[] a)	返回一个数组，该数组由当前 Set 中所有元素组成，返回数组的运行时类型由参数指定

例 5－2　TestSet.java：Set 容器的区别。

```
public class TestSet{
    public static void main(String[] args){
        Random rand=new Random(47);
        Set<Integer> s=new HashSet<Integer>();
        for(int i=0; i<5000; i++){
            s.add(rand.nextInt(40));
        }
        System.out.println(s);

        s=new TreeSet<Integer>();
        for(int i=0; i<5000; i++){
            s.add(rand.nextInt(40));
        }
        System.out.println(s);

        s=new LinkedHashSet<Integer>();
        for(int i=0; i<5000; i++){
            s.add(rand.nextInt(40));
        }
        System.out.println(s);
    }
}
```

```
/*一次执行的输出结果：
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 16, 19, 18, 21, 20, 23, 22, 25, 24, 27,
26, 29, 28, 31, 30, 34, 35, 32, 33, 38, 39, 36, 37]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
[28, 15, 39, 14, 16, 29, 24, 37, 6, 25, 2, 1, 32, 3, 36, 38, 5, 22, 23, 7, 8, 21, 33, 17, 18, 30,
31, 0, 27, 4, 19, 9, 12, 35, 34, 10, 26, 20, 13, 11]
*/
```

5.3 List接口及其实现

与数组类似地，List也可以建立数字索引与具体对象的关联，区别在于List可以自动扩充容量。Java SE中有两种类型的List：ArrayList和LinkedList。

表 5－2　List 的接口方法及其含义

接口方法	含　义
void add(int index，E element)	向当前 List 的指定位置插入特定元素
boolean addAll(int index，Collection<? extends E> c)	向当前 List 的指定位置插入一个容器中的所有元素
E get(int index)	返回当前 List 中指定位置的元素
int indexOf(Object o)	返回指定元素在当前 List 中第一次出现时的索引值，如果指定元素在当前 List 中不存在，则返回－1
int lastIndexOf(Object o)	返回指定元素在当前 List 中最后一次出现时的索引值，如果指定元素在当前 List 中不存在，则返回－1

续表

接口方法	含 义
E remove(int index)	移除当前 List 中指定位置的元素
E set(int index, E element)	用指定的元素替换当前 List 中指定位置的元素
List<E> subList(int fromIndex, int toIndex)	返回一个作为当前 List 一部分的子 List 的视图，子 List 的索引范围是[fromIndex, toIndex)。子视图不是一个新的 List，对子 List 的更改和删除将能反映到原 List

例 5－3　TestShapesArrayList.java：List 的基本用法。

```
public class TestShapesArrayList{
    public void draw(Shape s){
        s.draw();
    }
    public void draw(List<Shape> ss){
        for(int i=0; i<ss.size(); i++){
            draw(ss.get(i));
        }
    }
    public static void main(String[] args){
        List<Shape> ss=new ArrayList<Shape>();
        Random rand=new Random(37);
        for(int i=0; i<5; i++){
            switch(rand.nextInt(3)){
            case 0: ss.add(i, new Circle()); break;
            case 1: ss.add(new Rectangle()); break;
```

```
            case 2: ss.add(0, new Triangle()); break;
            default:
            }
        }
        TestShapesArrayList ts=new TestShapesArrayList();
        ts.draw(ss);

        System.out.println("=>Set");
        ss.set(ss.size()-2, new Triangle());
        ts.draw(ss);

        Shape s=ss.get(ss.size()-2);
        System.out.print("=>");
        s.erase();
        ss.remove(ss.indexOf(s));
        ts.draw(ss);
    }
```

```
}
class Shape {
    void draw() {}
    void erase() {}
}
class Circle extends Shape {
    void draw() { System.out.println("Draw Circle"); }
    void erase() { System.out.println("Erase Circle"); }
}
class Rectangle extends Shape {
    void draw() { System.out.println("Draw Rectangle"); }
    void erase() { System.out.println("Erase Rectangle"); }
}
class Triangle extends Shape {
    void draw() { System.out.println("Draw Triangle"); }
    void erase() { System.out.println("Erase Triangle"); }
}
```

```
/*一次执行的输出结果：
Draw Triangle
Draw Triangle
Draw Circle
Draw Circle
Draw Rectangle
=>Set
Draw Triangle
Draw Triangle
Draw Circle
Draw Triangle
Draw Rectangle
=>Erase Triangle
Draw Triangle
Draw Triangle
Draw Circle
Draw Rectangle
*/
```

5.4 Queue接口及其实现

Queue是一种先进先出(FIFO)的容器。程序从Queue的一端放入对象，从另一端取出对象，对象进入和取出的顺序是相同的。在Java SE中，LinkedList实现了Queue接口，因此LinkedList可以用作Queue的一种实现。

表5-3 Queue的接口方法及其含义

接口方法	含 义
boolean offer(E e)	向队列末尾加入指定元素，如果成功则返回true，否则返回false
boolean add(E e)	向队列末尾加入指定元素，如果加入失败则抛出异常
E peek()	获取但不删除队首元素(LinkedList的第一个元素)，如果队列为空，则返回null
E element()	获取但不删除队首元素(LinkedList的第一个元素)，如果队列为空，则抛出异常
E poll()	获取并删除队首元素(LinkedList的第一个元素)，如果队列为空，则返回null
E remove()	获取并删除队首元素(LinkedList的第一个元素)，如果队列为空，则抛出异常

例 5－4　TestQueue.java：Queue 的基本用法。

```
public class TestQueue{
    public static void printQueue(Queue q){
        while(q.peek()! =null)
            System.out.print(q.remove()+" ");
        System.out.println();
    }
    public static void main(String[] args){
        Queue<Integer> q=new LinkedList<Integer>();
        Random rand=new Random(37);
        for(int i=0; i<10; i++)
            q.offer(rand.nextInt(i+10));
        TestQueue.printQueue(q);

        Queue<Character> qc=new LinkedList<Character>();
        for(char c: "JavaLanguage".toCharArray())
            qc.offer(c);
        TestQueue.printQueue(qc);
    }
}
```

```
/*一次执行的输出结果：
5 4 6 11 2 6 7 9 11 2
J a v a L a n g u a g e
*/
```

例 5－5　TestPriorityQueue.java：优先队列的基本用法。

```
public class TestPriorityQueue{
    public static void printQueue(Queue q){
        while(q.peek()!=null)
            System.out.print(q.remove()+" ");
        System.out.println();
    }
    public static void main(String[] args){
        PriorityQueue<Integer> pq=new PriorityQueue<Integer>();
        Random rand=new Random(47);
        for(int i=0; i<10; i++)
            pq.offer(rand.nextInt(i+10));
        TestPriorityQueue.printQueue(pq);

        String fact="Java is a good programming language";
```

```
        List<String> strs=Arrays.asList(fact.split(""));
        PriorityQueue<String> strpq=new PriorityQueue<String>(strs);
        TestPriorityQueue.printQueue(strpq);

        strpq=new PriorityQueue<String>(strs.size(), Collections.reverseOrder());
        strpq.addAll(strs);
        TestPriorityQueue.printQueue(strpq);
    }
}
```

```
/*一次执行的输出结果:
0 1 1 1 1 1 3 5 8 14
            J a a a a a a d e g g g g g i i l m m n n o o o p r r s u v
v u s r r p o o o n n m m l i i g g g g g e d a a a a a a J
*/
```

5.5 Map接口及其实现

Map是一系列“键-值”之间的映射关系，是一种将对象(而非数字)与对象相关联的设计。在很多情况下，这一设计都能够有效地解决问题，例如，当我们需要统计一篇论文中的各个英文单词的个数时，可创立一个由String映射到Integer的Map。Map的值可以是复杂的结构，例如，当希望将一个科学家与

表 5－4 Map 的接口方法及其含义

接口方法	含 义
V put(K key, V value)	将指定的值 value 关联到当前 Map 中的指定的键 key
void putAll(Map<? extends K, ? extends V> m)	将参数 Map 中的所有键值对都添加到当前 Map 中
V get(Object key)	返回当前 Map 中指定的键所映射到的值，如果没有与该键相对应的映射，就返回 null
V remove(Object key)	移除当前 Map 中指定的键所对应的键值对
void clear()	从当前 Map 中移除所有键值对
boolean containsKey(Object key)	如果当前 Map 中存在从指定的键所发出的映射，则返回 true；否则返回 false
boolean containsValue(Object value)	如果当前 Map 中存在一个或多个键，能够映射到指定的值，那么返回 true；否则返回 false
boolean isEmpty()	如果当前 Map 中不含任何键值对，则返回 true；否则返回 false
Set<Map.Entry<K, V>> entrySet()	返回一个由当前 Map 中所有键值对所组成的 Set
Set<K> keySet()	返回一个 Set，其中包含了当前 Map 中的所有键
Collection<V> values()	返回一个 Collection，其中包含了当前 Map 中的所有值
int size()	返回当前 Map 中键值对的数量