第3章 Java集合框架

3.1 Java 集合概念

集合是什么呢？很难给集合下一个精确的定义，通常情况下，把具有相同性质的一类东西，汇聚成一个整体，就可以称为集合。比如某个学校的全体班级、某个公司的全体员工等都可以称为集合。了解 Java 中的集合，我们先从数据在内存中的存储结构说起，这样更容易理解集合。一般数据存储结构分为以下几种。

(1) 顺序存储：指元素在内存中连续的存储在一起，根据第一个元素的地址和每个元素所占的字节很容易计算其他任意位置的元素的地址，进而可以访问该元素，如数组。数组名就是数组的首地址，通过下标很快能够访问其他元素。这种存储方式有利于元素访问，但增加与删除元素的性能不高。大家应该有印象，对于一个已经开辟好空间的数组，如果删除其中某个位置元素的值，需要将其后的元素前移，再将最后元素赋予一个特殊的值，以示删除，其实本质上空间并未释放。在集合中有类 ArrayList 也是这样存储的。

(2) 链式存储：元素一般由值 Data 和 Next 域构成，元素在内存中不需要连续的空间，元素通过一个指向下一个元素地址的指针链在一起。这种存储的数据结构典型代表是单向链表。对于单向链表一般只需要保留链表的头指针，通过头指针依次取下一个元素可以访问任意位置的元素。在 Java 集合类中 LinkedList 是一个双向链表。

(3) 散列存储 Hash：元素值决定了对象在内存中的存储位置。元素值应该是唯一的。元素值本身并不能决定对象的存储位置，需要通过一种散列(Hash)技术来处理，产生一个被称作散列码(Hash code)的整数值，散列码通常用作一个偏置量，该偏置量是相对于分配给映射的内存区域起始位置的，由此确定元素的存储位置。理想情况下，散列处理应该产生给定范围内均匀分布的值，而且每个关键字应得到不同的散列码。在 Java 集合类中 HashSet 是一个使用 Hash 算法计算存储地址的类。

(4) Map 映射存储：与散列存储不同的是它每个元素由关键字(Key)与值(Value)构成，根据元素 Key 以及相应的散列算法计算元素存储地址。Key 不能重复，映射与集或列表有明显区别，映射中每个项都是成对的。内存中存储的每个元素都有一个相关的关键字对象。

集合类存放于 java.util 包中。集合类存放的都是对象的引用，而非对象本身，出于表达上的方便，我们称集合中的对象(或元素)就是指集合中对象的引用。

Java 集合框架主要由接口与其实现的类构成。图 3.1 描述了 Java 集合框架图。集合类型主要有 3 种：set(集)、list(列表)和 map(映射)。

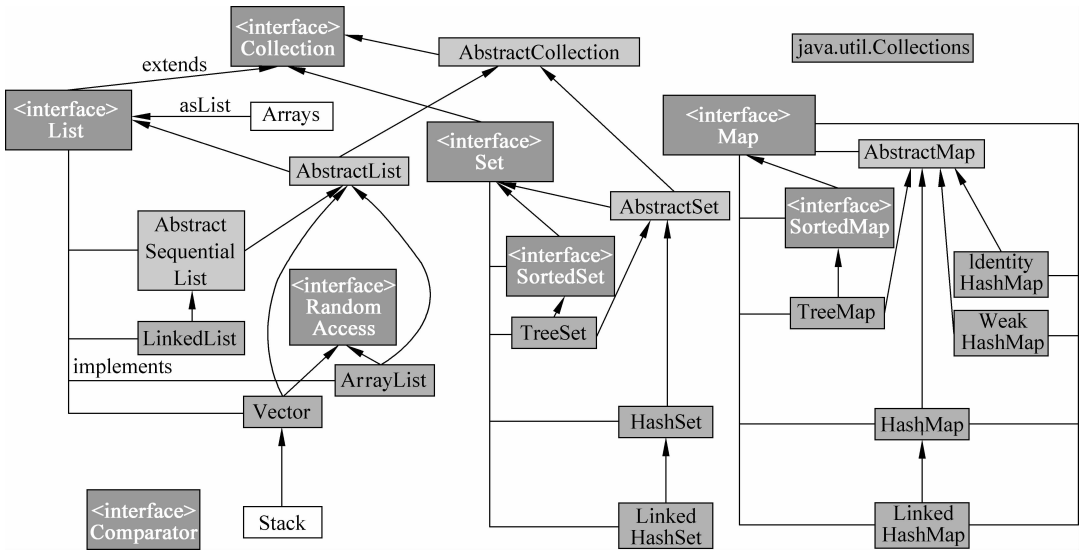

图 3.1 Java 集合框架图

1. 集 Set 接口

集是最简单的一种集合，Set 和数学中的集合是同一个概念，就是没有重复元素的集合，Set 接口不保证维护元素的次序。

HashSet 采用散列函数对元素进行排序，这是专门为快速查询而设计的；TreeSet 采用红黑树的数据结构进行排序元素；LinkedHashSet 内部使用散列以加快查询速度，同时使用链表维护元素的次序，使得元素看起来是以插入的顺序保存的。需要注意的是，生成自己的类时，Set 需要维护元素的存储顺序，因此要实现 Comparable 接口并定义 compareTo() 方法。

2. 列表 List 接口

列表的主要特征是其对象以线性方式存储，没有特定大小顺序，只有一个开头和一个结尾。当然，它与根本没有顺序的集是不同的。列表在数据结构中分别表现为数组和向量、链表、堆栈、队列。

3. 映射 Map 接口

映射与集或列表有明显区别，映射中每个项都是成对的。映射中存储的每个对象都有一个相关的关键字对象，关键字决定了对象在内存中的存储位置，检索对象时必须提供相应的关键字，就像在字典中查单词一样，关键字应该是唯一的。

3.2 Java 集合使用

3.2.1 HashSet 使用

1. HashSet 构造与增加元素

使用 HashSet set=new HashSet();就可以创建一个 HashSet 集合对象,可以向集合中添加任何对象,因为对象是传引用的,而集合中存放的就是对象的引用。集合中元素的存储空间是自动开辟的,不像数组需要预先开辟内存。

集合中元素是依据元素值和相应的哈希算法计算其地址,元素值相同地址就相同,值不同地址就不会相同。所以在 HashSet 集合中不存在元素值重复的元素。例如,HashSet 存储就像夜晚的星星一样排列,无所谓先后顺序。

```
HashSet set = new HashSet();
set.add("a");                          //向集合中添加一个 String
set.add(new Integer(1));               //向集合中添加一个 Integer
int x[ ] = {1,2,3,5};
set.add(x);                            //向集合中添加数组
Student s = new Student("1001","zhou");
set.add(s);                            //向集合中添加一个自定义类的对象
```

2. 遍历 HashSet 集合中的元素

集合中元素是依据元素值和相应的哈希算法计算其地址,所以如何读取集合中的元素,需要遍历算法即使用迭代器。所谓遍历是指按照某种顺序,对于集合中每个元素仅访问一次,不重复也不遗漏。Iterator(迭代)是指获取集合中元素的过程,实际上帮助获取集合中的元素。

迭代器代替了 Java Collections Framework 中的 Enumeration。迭代器与枚举不同之处在于迭代器在迭代期间可以从迭代器所指向的集合中移除元素。可以将迭代器想象为将集合中的散列的元素穿成一条线,可以沿着这条线访问从开始元素到最后的元素。

例 3.1 集合 HashSet 使用示例(HashSetDemo1.java)。

```
package chapter3;
import java.util.HashSet;
import java.util.Iterator;
public class HashSetDemo1 {
 public static void main(String[ ] args) {
       HashSet hs = new HashSet();
       hs.add(new String("Java"));
       hs.add(new String("c++"));
       hs.add(new Integer(100));
       hs.add(new Double(100.2));
       Iterator it = hs.iterator();
       while(it.hasNext()){              //判断是否还有下一个元素
```

```
            Object obj = it.next();        //取迭代器中下一个值
            if(obj instanceof String )     //判断是否是 String 类的实例
                System.out.println("String:" + obj);
            if(obj instanceof Integer )    //判断是否是 Integer 类的实例
                System.out.println("Integer:" + obj);
            if(obj instanceof Double )     //判断是否是 Double 类的实例
                System.out.println("Double:" + obj);
        }
    }
}
```

输出结果如图 3.2 所示。

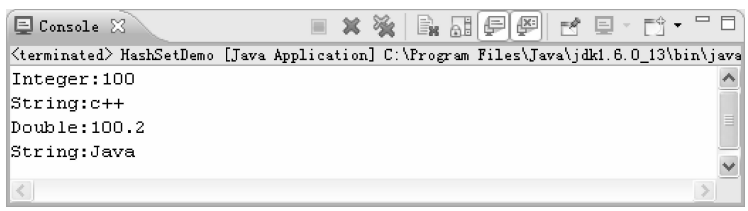

图 3.2 HashSetDemo1.java 输出结果

注意：

(1) 输出结果元素的顺序和实际放置的顺序不一致，事实上 Set 集合中不存在顺序的问题，其存放地址是计算出来的。

(2) 判断对象所属的类可以通过 obj.getClass().getName()得到类名来判断。

(3) 判断对象是否是数组 obj.getClass().isArray()。

3. 删除 HashSet 中元素

```
hs.remove(Object);
```

例如：

```
hs..remove("Java");                    //在集合 hs 中删除"Java"元素
```

删除所有元素可以使用集合对象的 clear()函数。

4. 判断是否包含某个元素 contains(Object)

例 3.2 集合 HashSet 使用示例(HashSetDemo2.java)。

```
package chapter3;
import java.util.HashSet;
public class HashSetDemo2 {
   public static void main(String[] args) {
        HashSet hs = new HashSet();
        hs.add(new String("Java"));
        hs.add(new String("c++"));
```

```
        hs.add(new Integer(100));
        hs.add(new Double(100.2));
        if(hs.contains(new String("Java")))
           System.out.println("集合中包含 java");
    }
}
```

输出结果如图 3.3 所示。

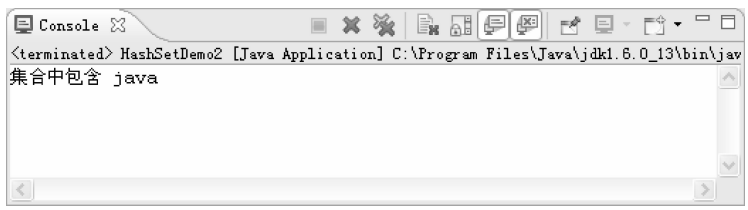

图 3.3 HashSetDemo2.java 输出结果

3.2.2 TreeSet 使用

TreeSet 是一个有序的 Set 集合,下面给出 TreeSet 使用示例。

例 3.3 集合 TreeSet 使用示例(TreeSetDemo1.java)。

```
package chapter3;
import java.util.Iterator;
import java.util.TreeSet;
public class TreeSetDemo1 {
public static void main(String[] argv) {
     TreeSet tree = new TreeSet();          //使用默认排序算法
     tree.add("d");                         //通过 add()方法向书中增加元素
     tree.add("c");
     tree.add("a");
     tree.add("b");
     Iterator it = tree.iterator();
     while (it.hasNext()) {
         System.out.print(it.next() + ",");
     }
    }
}
```

输出结果如图 3.4 所示。

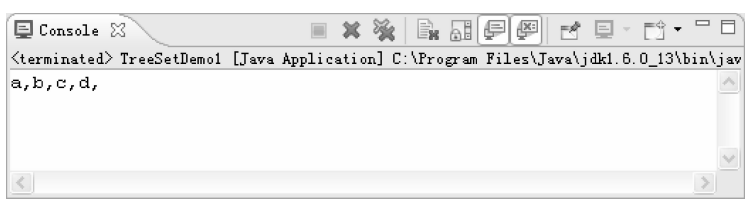

图 3.4 TreeSetDemo1.java 输出结果

从输出结果中可以看出，TreeSet 中的元素被排序了。只是该例子是使用默认的排序算法。如何自定义排序？可以在 Java 集合中实现接口 Comparator 来实现自定义排序，例如：

```
class MyCmp implements Comparator{
    public int compare(Object obj1,Object obj2){
         int x = 0;
         x = obj1.toString().compareTo(obj2.toString());
         return x;
    }
}
```

然后将该类的对象作为 TreeSet 构造函数的实际参数，即将自定义比较类对象作为构造函数参数，这样就可以使用自定义排序算法。该函数结果决定两个元素的先后位置。

```
TreeSet tree = new TreeSet(new MyCmp());
```

其原理是：向 tree 增加元素时，自动调用 MyCmp 类的 compare(obj1,obj2)函数，根据 compare(obj1,obj2)函数返回值决定元素 obj1、obj2 的顺序。

例 3.4 根据学生成绩排序，当成绩相同时按照学号进行排序。

(1) 定义学生类。

```
package chapter3.compare;
public class Student {
   private String sno, sname;
   private int score;
   public Student(String sno, String sname, int score) {
         super();
         this.sno = sno;
         this.sname = sname;
         this.score = score;
   }
   public String getSno() {
         return sno;
   }
   public void setSno(String sno) {
         this.sno = sno;
   }
   public String getSname() {
         return sname;
   }
   public void setSname(String sname) {
         this.sname = sname;
   }
   public int getScore() {
         return score;
   }
   public void setScore(int score) {
         this.score = score;
```

```
    }
}
```

(2) 定义一个用于比较的算法类。

```
package chapter3.compare;
import java.util.Comparator;
public class MyCmp implements Comparator {
    //向集合中添加元素时,自动调用 compare(Object obj1, Object obj2)
    //并将新增加的元素与原有元素比较,根据返回值决定新元素插入位置
    //示例中,根据学生成绩排序,当成绩相同时按照学号进行排序
    public int compare(Object obj1, Object obj2) {
        int x = 0;
        Student s1 = (Student)obj1;
        Student s2 = (Student)obj2;
        if(s1.getScore()>= s2.getScore())
            x = -1;
        else if(s1.getScore()< s2.getScore())
            x = 1;
        else{  //当成绩相等时比较学号
            x = s1.getSno().compareTo(s2.getSno());
        }
        return x;
    }
}
```

(3) 定义一个测试类。

```
package chapter3.compare;
import java.util.*;
public class Test {
   public static void main(String[] args) {
        TreeSet tree = new TreeSet(new MyCmp());
        Student s1 = new Student("1001","zhou",67);
        Student s2 = new Student("1002","lou",87);
        Student s3 = new Student("1003","zhang",87);
        Student s4 = new Student("1004","zhao",76);
        tree.add(s1);tree.add(s2);
        tree.add(s3);tree.add(s4);
        Iterator it = tree.iterator();
        while(it.hasNext()){
            Student s = (Student)it.next();
            System.out.println(s.getSno() + "," + s.getSname() + "," + s.getScore());
        }
   }
}
```

以下输出结果如图 3.5 所示。值得仔细研究一下,试着动手完成一个。

注意:compare(Object obj1, Object obj2)谁是新增元素,谁是集合中已有的元素? 请读者动手实验得出结论。

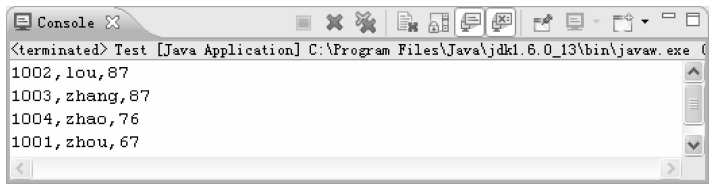

图 3.5 Test.java 输出结果

3.2.3 ArrayList 使用

接口 List 次序是 List 最重要的特点，它确保维护元素特定的顺序。List 从 Collection 接口继承过来并添加了一些抽象方法，例如，添加了插入与移除元素的抽象函数。

ArrayList 类实现 List 接口，可以将 ArrayList 理解成是一个动态的数组。既然它是一个数组，它就可以按照下标对其中元素进行操作。动态是指 ArrayList 能够自动分配或释放空间，它允许对元素进行快速随机访问，但是向 List 中间插入与移除元素的速度很慢。当元素的增加或移除发生在 List 中央位置时，效率很差。如果频繁地进行增加或删除元素操作，则不宜使用 ArrayList 类。

1. 构造与集合元素读写 ArrayList list = new ArrayList();

例 3.5 集合 ArrayList 使用示例(ArrayListDemo.java)。

```
package chapter3;
import java.util.*;
import chapter3.compare.Student;
public class ListDemo1 {
   public static void main(String[] argv) {
         List list = new ArrayList();
         Student s1 = new Student("1001","zhou",67);
         Student s2 = new Student("1002","lou",87);
         Student s3 = new Student("1003","zhang",87);
         Student s4 = new Student("1004","zhao",76);
         list.add(s1);list.add(s2);list.add(s3);list.add(s4);
         for(int i = 0;i < list.size();i++){
              Student s = (Student)list.get(i);
              System.out.println(s.getSno() + "," + s.getSname() + "," + s.getScore());
         }
   }
}
```

输出结果如图 3.6 所示。

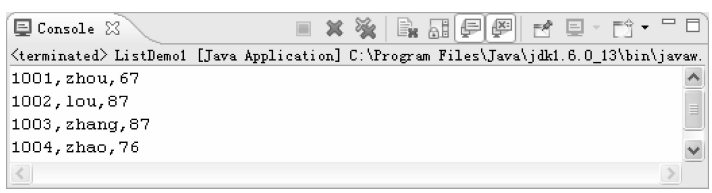

图 3.6 ListDemo1.java 输出结果

注意：ArrayList 是一个动态分配内存地址的数组，元素可以通过下标快速地访问，所以 ArrayList 要求连续的内存空间。可以通过索引或对象引用来删除 ArrayList 集合中的元素。例如，list.remove(2)；删除集合中第三个元素，也可以用 list.remove(s3)；来删除。

2. LinkedList 的使用

LinkedList 与 ArrayList 相反，适合用来进行增加和移除元素，但随机访问的速度较慢。此外，可以通过 LinkedList 来实现栈 stack 与队列 queue。

LinkedList 中的 addFirst()、addLast()、getFirst()、getLast()、removeFirst()、removeLast()等函数从前部、末尾或中间位置插入或删除元素。

例 3.6 集合 LinkedList 使用示例(LinkedListDemo.java)。

```
package chapter3;
import java.util.LinkedList;
public class LinkedListDemo {
   public static void main(String[] argv) {
         LinkedList list = new LinkedList();
         list.add("a");                                  //在尾部追加
         list.addFirst("b");                             //在头部增加元素
         list.addLast("c");                              //在尾部追加
         list.add(1, "d");                               //在第二个位置插入元素
         System.out.print(list.getFirst() + " ");        //读取头部元素
         System.out.print(list.getLast() + " ");         //读取尾部元素
         System.out.print(list.get(1) + " ");            //读第二个位置元素
         list.removeFirst();                             //删除头部元素
         list.removeLast();                              //删除尾部元素

         System.out.print(list.getFirst() + " ");        //读取头部元素
         System.out.print(list.getLast() + " ");         //读取尾部元素
   }
}
```

输出结果如图 3.7 所示。请仔细研究输出结果，注意 LinkList 的头部和尾部元素。

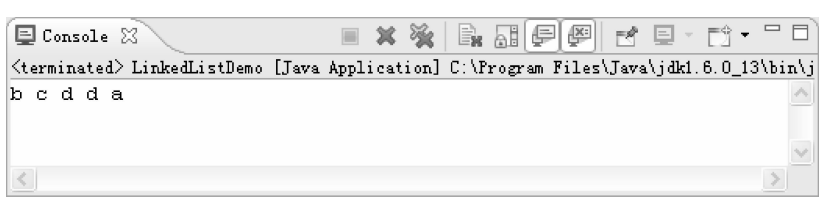

图 3.7 LinkedListDemo.java 输出结果

3.2.4 HashMap 使用

HashMap 是 Map 接口下的实现类，与 HashSet 不同的是它每个元素由关键字 Key 与值 Value 构成，根据元素 Key 以及相应的散列算法计算元素存储地址。

1. 构造 HashMap 以及向集合中添加元素

```
HashMap map = new HashMap();
map.put(key, value);
map.put("1001", "zhou");            //向 map 中添加元素,第一个参数为 key,第二个参数为 value
map.put("1002", "zhang");
map.put("1003", "zhou");
```

2. 在 Map 接口中可以根据关键字查找对应元素值

```
String s = map.get("1002").toString(); //"1002"为 key,该函数返回关键字对应的值
```

3. 如何遍历集合中所有元素

有两种思路：第一种是读出集合中所有的关键字，根据关键字集合依次查找各个元素的值。第二种是能不能把 Map 看成是 Set 一样，只是 Map 集合中元素由两个对象组成，可以把这两个对象看成一个对象的两个属性，然后就可以遍历了。

例 3.7 第一种遍历方法示例(MapDemo1.java)。

```
package chapter3;
import java.util.*;
public class MapDemo1 {
public static void main(String[] args) {
     HashMap map = new HashMap();
     map.put("1001", "张军");
     map.put("1002", "李元");
     map.put("1003", "王钧");
     Set keys = map.keySet();                          //读取所有关键字集合
     Iterator it = keys.iterator();                    //遍历关键字集合
     while (it.hasNext()) {
          String s = map.get(it.next()).toString(); //通过关键字查找元素值
          System.out.println(s);
     }
   }
}
```

输出结果如图 3.8 所示。

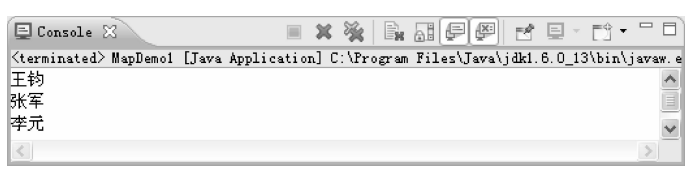

图 3.8 MapDemo1.java 输出结果

例 3.8 第二种遍历方法示例(MapDemo2.java)。

```
package chapter3;
import java.util.*;
```