


第10章

Java 集合类

( 视频讲解：45 分钟)

集合可以看作是一个容器，如红色的衣服可以看作是一个集合，所有 Java 类的书可以看作是一个集合。对于集合中的各个对象很容易将其存放到集合中，也很容易将其从集合中取出来，还可以将其按照一定的顺序进行摆放。Java 中提供了不同的集合类，这些类具有不同的存储对象的方式，并提供了相应的方法方便用户对集合进行遍历、添加、删除以及查找指定的对象。学习 Java 语言一定要学会使用集合。本章向读者介绍 Java 中的各种集合类。

通过阅读本章，您可以：

- » 掌握集合类接口的常用方法
- » 掌握集合类接口的实现类
- » 掌握迭代器的创建和使用



10.1 集合类概述

 视频讲解：光盘\TM\lx\10\集合类概述.exe

Java 语言的 `java.util` 包中提供了一些集合类，这些集合类又被称为容器。提到容器不难会想到数组，集合类与数组的不同之处是，数组的长度是固定的，集合的长度是可变的；数组用来存放基本类型的数据，集合用来存放对象的引用。常用的集合有 List 集合、Set 集合、Map 集合，其中 List 与 Set 实现了 Collection 接口。各接口还提供了不同的实现类。上述集合类的继承关系如图 10.1 所示。

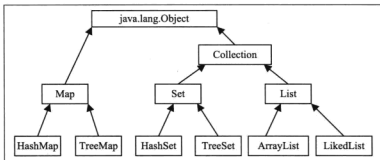


图 10.1 常用集合类的继承关系

10.2 集合类接口的常用方法

 视频讲解：光盘\TM\lx\10\集合类接口的常用方法.exe

Collection 接口是层次结构中的根接口。构成 Collection 的单位，被称之为元素。Collection 接口通常不能直接使用，但该接口提供了添加和删除元素、管理数据的方法。由于 List 接口与 Set 接口都实现了 Collection 接口，因此这些方法对 List 集合与 Set 集合是通用的。

10.2.1 List 接口的常用方法

List 接口继承了 Collection 接口，因此包含 Collection 中的所有方法。又因为 List 是列表类型，所以 List 接口还提供了一些适合于自身的常用方法，如表 10.1 所示。

表 10.1 List 接口的常用方法

方 法	返 回 值	功 能 描 述
<code>add(int index, Object obj)</code>	<code>void</code>	用来向集合中的指定索引位置添加对象，集合的索引位置从 0 开始，其他对象的索引位置相对向后移一位

续表

方 法	返 回 值	功 能 描 述
addAll(int index, Collection coll)	boolean	向集合的指定索引位置添加指定的集合对象
remove(int index)	Object	用来移除集合中指定索引位置的元素
get(int index)	Object	用于获取指定索引位置的元素
indexOf(Object obj)	int	该方法返回列表中对象第一次出现的索引位置, 如果集合中不包含该元素, 则返回-1
lastIndexOf(Object obj)	int	该方法返回列表中对象最后一次出现的索引位置, 如果集合中不包含该元素, 则返回-1
subList(int formIndex, int toIndex)	List	获取从索引 formIndex 到 toIndex 之间的元素对象
set(int index, E element)	Object	用指定元素替换列表中指定位置的元素, 返回以前在指定位置的元素
listIterator()	ListIterator	用来获得一个包含所有对象的 ListIterator 列表迭代器

从表 10.1 中可以看出, List 接口中适合于自身的方法都与索引有关。由于 List 集合以线性方式存储对象, 因此可以通过对象的索引来操作对象。

在 List 集合的众多方法中, add(int index, Object obj)方法与 set(int index, Object obj)方法不容易区分。通过下面的实例, 可以看出这两个方法之间的差别。

【例 10.1】 创建集合对象, 并向集合中添加元素, 通过 set()方法修改集合中元素, 再通过 add()方法向集合中添加元素, 通过迭代器遍历集合元素。(实例位置: 光盘\TM\10\1)

```
public class CollectionDemo {
    public static void main(String[] args) {
        String a = "A", b = "B", c = "C", d = "D", e = "E";           //定义要插入集合的字符串对象
        List<String> list = new LinkedList<String>();                 //创建 List 集合
        list.add(a);                                                    //向集合中添加元素
        list.add(e);
        list.add(d);
        Iterator<String> fristIterator = list.iterator();              //创建集合的迭代器
        System.out.println("修改前集合中的元素是: ");                //输出信息
        while (fristIterator.hasNext()) {                               //遍历集合中元素
            System.out.print(fristIterator.next() + " ");
        }
        list.set(1, b);                                                  //将索引位置为 1 的对象修改为对象 b
        list.add(2, c);                                                  //将对象 c 添加到索引位置为 2 的位置
        Iterator<String> it = list.iterator();                          //创建将集合对象修改之后的迭代器对象
        System.out.println();
        System.out.println("修改后集合中的元素是: ");
        while (it.hasNext()) {                                          //循环获取集合中元素
            System.out.print(it.next() + " ");
        }
    }
}
```

运行结果如图 10.2 所示。

List 集合中可以包含重复的对象, 若要获取重复对象第一次出现的索引位置可以使用 indexOf()方

法，想要获取重复对象最后一次出现的索引位置，可以使用 `lastIndexOf()` 方法。使用 `indexOf()` 与 `lastIndexOf()` 方法时，如果指定的对象在 `List` 集合中只有一个，则通过这两个方法获得的索引位置是相同的。

【例 10.2】 创建 `List` 集合对象，通过 `add()` 方法向集合中添加元素，并将对象 `apple`、`b` 在集合中第一次与最后一次出现的索引位置输出。（实例位置：光盘\TM\sl\10\2）

```
public class CollectionDemo {
    public static void main(String[] args) {
        String a = "a", b = "b", c = "c", d = "d", apple = "apple"; //要添加到集合中的对象
        List<String> list = new ArrayList<String>(); //创建 List 集合对象
        list.add(a); //对象 a 的索引位置为 0
        list.add(apple); //对象 apple 的索引位置为 1
        list.add(b); //对象 b 的索引位置为 2
        list.add(apple); //对象 apple 的索引位置为 3
        list.add(c); //对象 c 的索引位置为 4
        list.add(apple); //对象 apple 的索引位置为 5
        list.add(d); //对象 d 的索引位置为 6
        System.out.println(list); //输出列表中的全部元素
        System.out.println("apple 第一次出现的索引位置是: "+list.indexOf(apple));
        System.out.println("apple 最后一次出现的索引位置是: "+list.lastIndexOf(apple));
        System.out.println("b 第一次出现的索引位置是: "+list.indexOf(b));
        System.out.println("b 最后一次出现的索引位置是: "+list.lastIndexOf(b));
    }
}
```

运行结果如图 10.3 所示。

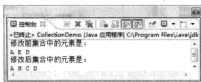


图 10.2 修改集合中的元素

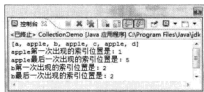


图 10.3 实例运行结果

10.2.2 Set 接口的常用方法

`Set` 集合由 `Set` 接口和 `Set` 接口的实现类组成。`Set` 接口继承了 `Collection` 接口，因此包含 `Collection` 接口的所有方法。`Set` 接口中的常用方法如表 10.2 所示。

表 10.2 Set 接口中的常用方法

方 法	返 回 值	功 能 描 述
<code>add(Object obj)</code>	<code>boolean</code>	如果此 <code>Set</code> 集合中尚未存在指定的元素，则添加此元素
<code>addAll(Collection coll)</code>	<code>boolean</code>	将参数集合中所有元素添加到此 <code>Set</code> 集合的尾部
<code>remove(Object obj)</code>	<code>boolean</code>	将指定的参数对象移除集合

续表

方 法	返 回 值	功 能 描 述
retainAll(Collection c)	boolean	只保存 Set 集合中包含在指定 Collection 集合中的内容
removeAll(Collection c)	boolean	在 Set 集合中移除包含在指定 Collection 中的元素
clear()	void	移除此 Set 中的所有元素
iterator()	Iterator	返回此 Set 中的元素上进行迭代的迭代器
size()	int	返回此 Set 集合中的所有元素数
isEmpty()	boolean	如果 Set 不包含元素, 则返回 true

由于 Set 集合中不允许存在重复值, 因此可以使用 Set 集合中的 addAll()方法, 将 Collection 集合添加到 Set 集合中并除掉重复值。

【例 10.3】 创建一个 List 集合对象, 并往 List 集合中添加元素。再创建一个 Set 集合, 利用 addAll()方法把 List 集合对象存入到 Set 集合中并除掉重复值, 最后打印 Set 集合中的元素。(实例位置: 光盘\TM\10\3)

```
public class CollectionDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();           //创建 List 集合对象
        list.add("apple");                                     //向集合中添加元素
        list.add("pear");
        list.add("banana");
        list.add("apple");
        Set<String> set = new HashSet<String>();              //创建 Set 集合对象
        set.addAll(list);                                     //将 List 集合添加到 Set 集合中
        Iterator<String> it = set.iterator();                 //创建 Set 集合迭代器
        System.out.println("集合中的元素是: ");
        while (it.hasNext()) {
            System.out.print(it.next()+"\t");
        }
    }
}
```

运行结果如图 10.4 所示。

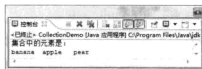


图 10.4 输出集合中的元素

10.2.3 Map 接口的常用方法

Map 接口提供了将键映射到值的对象。一个映射不能包含重复的键; 每个键最多只能映射到一个值。Map 接口中同样提供了集合的常用方法, 如 clear()、isEmpty()、size()等, 除此之外还包括如表 10.3 所示的常用方法。

表 10.3 Map 接口中的常用方法

方 法	返 回 值	功 能 描 述
put(key k,value v)	Object	向集合中添加指定的 key 与 value 的映射关系
containsKey(Object key)	boolean	如果此映射包含指定键的映射关系, 则返回 true

方 法	返 回 值	功 能 描 述
containsValue(Object value)	boolean	如果此映射将一个或多个键映射到指定值，则返回 true
get(Object key)	Object	如果存在指定的键对象，则返回该对象对应的值，否则返回 null
keySet()	Set	返回该集中的所有键对象组成的 Set 集合
values()	Collection	返回该集中所有值对象形成的 Collection 集合

由于 Map 集合中的元素是通过 key、value 进行存储的，要获取集合中指定的 key 值或 value 值，需要先通过相应的方法获取 key 集合或 value 集合，再遍历 key 集合或 value 集合获取指定值。

【例 10.4】 向一个 Map 集合中插入元素并根据 key 的值打印集合中的元素。（实例位置：光盘\TM\sh10\4）

```
public class MapDemo {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<String, String>();           //创建 Map 集合
        map.put("1", "apple");                                              //向集合中添加对象
        map.put("2", "pear");
        map.put("3", "orange");
        for (int i = 1; i <= 3; i++) {
            System.out.println("第"+i+"元素是: "+map.get(i + ""));        //输出对应位置的元素值
        }
    }
}
```

运行结果如图 10.5 所示。



图 10.5 向 Map 集合中插入元素

10.2.4 范例 1：用 List 集合传递学生信息

集合在程序开发中经常用到，如在业务方法中将学生信息、商品信息等存储到集合中，然后作为方法的返回值返回给调用者，以此传递大量有序数据。本范例将使用 List 集合在方法之间传递学生信息。运行结果如图 10.6 所示。（实例位置：光盘\TM\sh10\5）

（1）在项目中新建窗体类 ClassInfo。在窗体中添加滚动面板，这个面板将放置表格控件。代码如下：

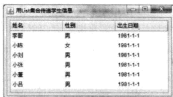


图 10.6 将学生信息添加到集合中

```

public ClassInfo() {
    setTitle("\u7528List\u96C6\u5408\u4F20\u9012\u5B66\u751F\u4FE1\u606F");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 392, 223);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JScrollPane scrollPane = new JScrollPane();
    contentPane.add(scrollPane, BorderLayout.CENTER);
    scrollPane.setViewportView(getTable());
}

```

(2) 编写 `getTable()` 方法, 在该方法中创建表格对象, 设置表格的数据模型, 然后调用 `getStudents()` 方法获取保存学生信息的集合对象, 在遍历该集合对象的同时把每个元素添加到表格模型的行, 并显示到表格控件中。代码如下:

```

private JTable getTable() {
    if (table == null) {
        table = new JTable(); //创建表格控件
        table.setRowHeight(23); //设置行高度
        String[] columns={"姓名","性别","出生日期"}; //创建列名数组
        DefaultTableModel model=new DefaultTableModel(columns,0); //创建表格模型
        table.setModel(model); //设置表格模型
        List<String> students = getStudents(); //调用方法传递 List 集合对象
        for (String info : students) { //遍历学生集合对象
            String[] args = info.split(","); //把学生信息拆分为数组
            model.addRow(args); //把学生信息添加到表格的行
        }
    }
    return table;
}

```

(3) 编写 `getStudents()` 方法, 该方法将向调用者传递 `List` 集合对象, 方法中为集合对象添加了多个元素, 每个元素值都是一个学生信息, 其中包括姓名、性别、出生日期。代码如下:

```

private List<String> getStudents() {
    List<String> list=new ArrayList<String>(); //创建 List 集合对象
    list.add("李哥,男,1981-1-1"); //添加数据到集合对象
    list.add("小陈,女,1981-1-1");
    list.add("小刘,男,1981-1-1");
    list.add("小张,男,1981-1-1");
    list.add("小董,男,1981-1-1");
    list.add("小吕,男,1981-1-1");
    return list;
}

```

10.2.5 范例 2: Map 集合二级联动

Map 集合可以保存键值映射关系，这非常适合本范例所需要的数据结构，所有省份信息可以保存为 Map 集合的键，而每个键可以保存对应的城市信息，本范例就利用这个 Map 集合实现了省市级联选择框，当选择省份信息时，将改变城市下拉列表框对应的内容。运行结果如图 10.7 所示。（实例位置：光盘\TM\sl\10\6）

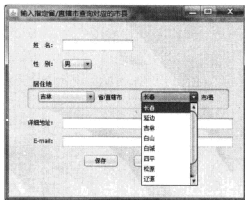


图 10.7 省市级联选择框

(1) 在项目中新建窗体类 CityMap，在该类中创建并初始化 Map 集合对象，在该集合对象中保存各省市的关联信息。代码如下：

```
public class CityMap {
    public static Map<String,String[]> model=new LinkedHashMap();
    static{
        model.put("北京", new String[]{"北京"});
        model.put("上海", new String[]{"上海"});
        model.put("天津", new String[]{"天津"});
        model.put("重庆", new String[]{"重庆"});
        model.put("黑龙江", new String[]{"哈尔滨","齐齐哈尔","牡丹江","大庆","伊春","双鸭山","鹤岗","鸡西","佳木斯","七台河","黑河","绥化","大兴安岭"});
        model.put("吉林", new String[]{"长春","延边","吉林","白山","白城","四平","松原","辽源","大安","通化"});
        model.put("辽宁", new String[]{"沈阳","大连","葫芦岛","旅顺","本溪","抚顺","铁岭","辽阳","营口","阜新","朝阳","锦州","丹东","鞍山"});
        //省略类似代码
    }
}
```

(2) 创建 MainFrame 主窗体类，在该类中添加 3 个文本框用于输入姓名、详细地址和 E-mail 信息，添加选择性别下拉列表框，添加“保存”和“重置”按钮，最后添加两个核心控件，也就是选

择省份与选择城市的下拉列表框。

(3) 编写 `getProvince()` 方法，在该方法中获取 `Map` 集合的键映射，也就是省份信息的 `Set` 集合，然后将该集合转换为数组，并作为方法的返回值，这个方法将在省份下拉列表框的初始化代码中调用。代码如下：

```
public Object[] getProvince() {
    Map<String, String[]> map = CityMap.model;           //获取省份信息保存到 Map 中
    Set<String> set = map.keySet();                       //获取 Map 集合中的键，并以 Set 集合返回
    Object[] province = set.toArray();                     //转换为数组
    return province;                                       //返回获取的省份信息
}
```

(4) 编写选择省份的下拉列表框的事件处理方法，该方法在省份下拉列表框改变选项时被调用，方法首先获取下拉列表框的选项值，然后把该值作为键并到 `Map` 集合中查找对应该键的值，返回结果是对应省份的所有城市名称组成的数组，最后用这个数组创建一个数据模型添加到城市下拉列表框控件中，以更新内容。代码如下：

```
private void itemChange() {
    String selectProvince = (String) comboBox.getSelectedItemAt();
    cityComboBox.removeAllItems();                          //清空市/县列表
    String[] arrCity = getCity(selectProvince);              //获取市/县
    cityComboBox.setModel(new DefaultComboBoxModel(arrCity)); //重新添加市/县列表的值
}
```

(5) 编写 `getCity()` 方法，该方法主要负责获取对应省份的城市数组，它将在省份下拉列表框控件的事件处理方法中被调用。代码如下：

```
public String[] getCity(String selectProvince) {
    Map<String, String[]> map = CityMap.model;           //获取省份信息保存到 Map 中
    String[] arrCity = map.get(selectProvince);           //获取指定键的值
    return arrCity;                                       //返回获取的市/县
}
```

10.3 集合类接口的实现类

 视频讲解：光盘\TM\lx\10\集合类接口的实现类.exe

10.3.1 List 接口的实现类

要使用 `List` 集合，通常情况下需要声明为 `List` 类型，然后通过 `List` 接口的实现类来对集合进行实例化。`List` 接口的实现类常用的有 `ArrayList` 与 `LinkedList`。

1. ArrayList 类

该类实现了可变的数组，允许所有元素，包括 `null`。可以根据索引位置对集合进行快速的随机访问。缺点是向指定的索引位置插入对象或删除对象的速度较慢。语法格式如下：

```
List<String> list = new ArrayList<String>();
```

2. LinkedList 类

该类采用链表结构保存对象。这种结构的优点是便于向集合中插入和删除对象，经常需要向集合中插入、删除对象时，使用 `LinkedList` 类实现的 `List` 集合的效率较好；但对于随机访问集合中的对象，使用 `LinkedList` 类实现 `List` 集合的效率较慢。语法格式如下：

```
List<String> list2 = new LinkedList<String>();
```

使用 `List` 集合时通常声明为 `List` 类型，可通过不同的实现类来实例化集合。

【例 10.5】 分别通过 `ArrayList`、`LinkedList` 类实例化 `List` 集合。

```
List list = new ArrayList();
```

```
List list2 = new LinkedList();
```

【例 10.6】 在项目中创建 `Gather` 类，在主方法中创建集合对象，通过 `Math` 类的 `random()` 方法随机获取集合中的某个元素，然后移除数组中索引位置为 2 的元素，最后遍历数组。（实例位置：光盘\TM\sl\10\7）

```
public class Gather {
    public static void main(String[] args) {
        List list = new ArrayList();
        int i = (int) (Math.random() * (list.size()-1));
        list.add("a");
        list.add("b");
        list.add("c");
        System.out.println("随机获取数组中的元素: "+list.get(i));
        list.remove(2);
        System.out.println("将索引是'2'的元素从数组移除后，数组中的元素是:");
        for (int j = 0; j < list.size(); j++) {
            System.out.print(list.get(j)+" ");
        }
    }
}
```

//创建 Gather 类
//主方法
//创建集合对象
//获得 0~2 之间的随机数
//向集合中添加元素

//将指定索引位置的元素从集合中移除
//循环遍历集合

运行结果如图 10.8 所示。

10.3.2 Set 接口的实现类

要使用 `Set` 集合，通常情况下需要声明为 `Set` 类型，然后

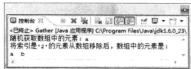


图 10.8 移除集合中索引为 2 的元素

通过 Set 接口的实现类来实例化。Set 接口的实现类常用的有 HashSet 和 TreeSet 类。语法格式如下：

```
Set<String> collSet = new HashSet<String>();
Set<String> collSet2 = new TreeSet<String>();
```

由于 Set 集合中的对象是无序的，遍历 Set 集合的结果与插入 Set 集合的顺序并不相同。

【例 10.7】 遍历输出 HashSet 中的全部元素。（实例位置：光盘\TM\sl\10\8）

（1）创建 People 对象，该对象中包含有 String 类型与 long 类型属性，并包含有属性的 setXXX() 与 getXXX() 方法。代码如下：

```
public class People {
    private String name;
    private long id_card;
    public People(String name, long id_card) {
        this.name = name;
        this.id_card = id_card;
    }
    public long getId_card() {
        return id_card;
    }
    public void setId_card(long id_card) {
        this.id_card = id_card;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

（2）在 main() 方法中，创建 Set 集合对象，并向集合中添加对象，通过迭代器将集合中的元素输出。代码如下：

```
public class CollectionDemo {
    public static void main(String[] args) {
        Set<People> hashSet = new HashSet<People>(); //创建 Set 集合对象
        hashSet.add(new People("陈同学", 201101)); //向集合中添加对象
        hashSet.add(new People("王同学", 201102));
        hashSet.add(new People("李同学", 201103));
        Iterator<People> it = hashSet.iterator(); //创建集合迭代器
        System.out.println("集合中的元素是：");
        while (it.hasNext()) { //循环遍历迭代器
            People person = it.next();
            System.out.println(person.getName() + " " + person.getId_card());
        }
    }
}
```

运行结果如图 10.9 所示。

10.3.3 Map 接口的实现类

Map 接口常用的实现类有 HashMap 和 TreeMap。通常建议使用 HashMap 实现类实现 Map 集合，因为由 HashMap 类实现的 Map 集合对于添加和删除映射关系效率更高。HashMap 是基于哈希表的 Map 接口的实现，HashMap 通过哈希码对其内部的映射关系进行快速查找；由 HashMap 类实现的 Map 集合对于添加或删除映射关系效率较高；而 TreeMap 中的映射关系存在一定的顺序，如果希望 Map 集合中的对象存在一定的顺序，应该使用 TreeMap 类实现 Map 集合。

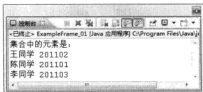


图 10.9 输出集合中所有的元素

1. HashMap 类

该类基于哈希表的 Map 接口的实现，此实现提供所有可选的映射操作，并允许使用 null 值和 null 键，但必须保证键的唯一性。HashMap 通过哈希码对其内部的映射关系进行快速查找。此类不保证映射的顺序，特别是不保证该顺序恒久不变。

2. TreeMap 类

该类不仅实现了 Map 接口，还实现了 java.util.SortedMap 接口，因此集合中的映射关系具有一定的顺序。但在添加、删除和定位映射关系上，TreeMap 类比 HashMap 类的性能差一些。由于 TreeMap 类实现的 Map 集合中的映射关系是根据键对象按照一定的顺序排列的，因此不允许键对象是 null。

可以通过 HashMap 类创建 Map 集合，当需要顺序输出时，再创建一个完成相同映射关系的 TreeMap 类实例。

【例 10.8】 通过 HashMap 类实例化 Map 集合，并遍历该 Map 集合，然后创建 TreeMap 实例实现将集合中的元素顺序输出。（实例位置：光盘\TM\sl\10\9）

（1）首先创建 Emp 类，代码如下：

```
public class Emp {
    private String e_id;
    private String e_name;
    public Emp( String e_id,String e_name){
        this.e_id = e_id;
        this.e_name = e_name;
    }
    /*****省略了属性的 setXXX()以及 getXXX()方法*****/
}
```

（2）创建一个用于测试的主类。首先新建一个 Map 集合，并添加集合对象，分别遍历由 HashMap 与 TreeMap 类实现的 Map 集合，观察两者的不同点。代码如下：

```
public class MapText {
    public static void main(String[] args) {
        //创建 MapText 类
        //主方法
    }
}
```

```

Map map = new HashMap(); //由 HashMap 实现的 Map 对象
Emp emp = new Emp("001", "张三");
Emp emp2 = new Emp("005", "李四");
Emp emp3 = new Emp("004", "王一"); //创建 Emp 对象
map.put(emp.getId(), emp.getName());
map.put(emp2.getId(), emp2.getName()); //将对象添加到集合中
map.put(emp3.getId(), emp3.getName());
Set set = map.keySet(); //获取 Map 集合中的 key 对象集合
Iterator it = set.iterator();
System.out.println("HashMap 类实现的 Map 集合, 无序: ");
while (it.hasNext()) {
    String str = (String) it.next();
    String name = (String) map.get(str);
    System.out.println(str + " " + name);
} //遍历 Map 集合

TreeMap treemap = new TreeMap(); //创建 TreeMap 集合对象
treemap.putAll(map); //向集合添加对象
Iterator iter = treemap.keySet().iterator();
System.out.println("TreeMap 类实现的 Map 集合, 键对象升序: ");
while (iter.hasNext()) {
    String str = (String) iter.next(); //遍历 TreeMap 集合对象
    String name = (String) map.get(str); //获取集合中的所有 key 对象
    System.out.println(str + " " + name); //获取集合中的所有 values 值
}
}
}

```

运行结果如图 10.10 所示。

10.3.4 范例 3: for 循环遍历 ArrayList

在使用集合类时, 不仅要了解容器是如何保存元素的, 而且要知道如何取出元素。本范例将使用普通 for 循环遍历 ArrayList, 从中取出所有序号为奇数的元素。运行结果如图 10.11 所示。(实例位置: 光盘\TM\sl\10\10)



图 10.10 输出 Map 集合中的元素

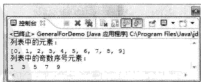


图 10.11 输出序号为奇数的元素

在项目中新建 GeneralForDemo 类, 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型,

并为其添加 10 个元素，利用 for 循环遍历 ArrayList 集合，输出列表中序号为奇数的元素。代码如下：

```
public class GeneralForDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();           //创建列表
        for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的元素: " + list);           //输出列表中全部的元素
        System.out.println("列表中的奇数序号元素: ");
        for (int i = 1; i < list.size(); i += 2) {              //输出列表中序号为奇数的元素
            System.out.print(list.get(i) + " ");
        }
    }
}
```

10.3.5 范例 4：用动态数组保存学生姓名

Java 语言中提供了各种数据集合类，这些类主要用于保存复杂结构的数据，其中 ArrayList 集合可以看作动态数组。它突破普通数组固定长度的限制，可以随时向数组中添加和移除元素，这将使数组更加灵活，如果要获取普通数组，还可以通过该类的 toArray() 方法获得。本范例通过这个 ArrayList 集合类实现向程序动态添加与删除学生姓名的功能，其中所有数据都保存在 ArrayList 集合的实例对象中。运行结果如图 10.12 所示。（实例位置：光盘\TM\10\11）

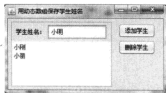


图 10.12 用动态数组保存学生姓名

（1）在项目中新建 DynamicArray 窗体类，在窗体中添加文本框控件、列表控件、“添加学生”按钮和“删除学生”按钮。编写“添加学生”按钮的事件处理方法，在该方法中获取用户在文本框中输入的字符串，然后将这个字符串添加到 ArrayList 集合中，再调用 replaceModel() 方法把集合中的数据显示到窗体的列表控件中。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    textField.requestFocusInWindow();
    textField.selectAll();                                     //选择文本框中的文本准备下次输入
    String text = textField.getText();                       //获取用户输入的姓名
    if (text.isEmpty())                                     //过滤为输入姓名的情况
        return;
    arraylist.add(text);                                     //把姓名添加到数组集合中
    replaceModel();                                         //把数组集合中的内容显示到界面列表控件中
}
```

（2）编写“删除学生”按钮的事件处理方法，在该方法中获取列表控件的当前选择项，然后从

ArrayList 集合中移除这个选项的值，最后调用 `replaceModel()` 方法把集合中的数据显示到窗体的列表控件中。代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    Object value = list.getSelectedValue();           //获取列表控件的选择项
    arraylist.remove(value);                         //从数组集合中移除用户的选择项
    replaceModel();                                 //把数组集合中的内容显示到界面列表控件中
}
```

(3) 编写 `replaceModel()` 方法，在该方法中重新设置列表控件的模型，模型要读取 ArrayList 集合的元素并显示到列表控件中。代码如下：

```
private void replaceModel() {
    list.setModel(new AbstractListModel() {          //为列表控件设置数据模型并显示数组集合中的数据
        @Override
        public int getSize() {                      //获取数组大小
            return arraylist.size();
        }
        @Override
        public Object getElementAt(int index) {      //获取指定索引元素
            return arraylist.get(index);
        }
    });
}
```

10.4 迭 代 器

 视频讲解：光盘\TM\lx\10\迭代器.exe

10.4.1 迭代器的创建和使用

利用 Iterator 接口创建迭代器，Iterator 接口位于 `java.util` 包下。Iterator 接口中有 3 个方法。

- ☑ `hasNext()`：如果仍有元素可以迭代，则返回 `true`。
- ☑ `next()`：返回迭代的下一个元素。
- ☑ `remove()`：从迭代器指向的 collection 中移除迭代器返回的最后一个元素（可选操作）。

ListIterator 接口继承了 Iterator 接口，下面通过一个简单的实例演示迭代器的用法。

【例 10.9】 创建 `ListIteratorDemo` 类，在 `main()` 方法中创建了一个 ArrayList 对象，然后打印集合中的元素和位置等信息。（实例位置：光盘\TM\lx\10\12）

```
public class ListIteratorDemo {
    public static void main(String[] args) {
        //创建一个 ArrayList，其中能够保存的数据类型是 Integer
        ArrayList<Integer> array = new ArrayList<Integer>();
```

```

//使用 Collections 类中的工具方法 addAll()向集合中增加元素
Collections.addAll(array, 1, 2, 3, 4, 5, 6);
System.out.println("集合中的元素: " + array);
//使用无参数的方法获得 ListIterator 对象
ListIterator<Integer> iterator = array.listIterator();
//对于初始位置, 判断是否具有下一个元素
boolean hasNext = iterator.hasNext();
System.out.println("集合是否具有下一个元素: " + hasNext);
//对于初始位置, 判断是否具有前一个元素
boolean hasPrevious = iterator.hasPrevious();
System.out.println("集合是否具有前一个元素: " + hasPrevious);
int next = iterator.next();
//获得下一个元素
System.out.println("获得集合的下一个元素: " + next);
int nextIndex = iterator.nextIndex();
//获得下一个元素的索引
System.out.println("获得集合的下一个元素的索引: " + nextIndex);
int previous = iterator.previous();
//获得前一个元素
System.out.println("获得集合的前一个元素: " + previous);
int previousIndex = iterator.previousIndex();
//获得前一个元素的索引
System.out.println("获得集合的前一个元素的索引: " + previousIndex);
iterator.add(7);
//向列表中增加元素 7
iterator.next();
//获得下一个元素
iterator.set(12);
//将获得的元素设置成 12
System.out.println("将获得的下一个元素修改成 12 后的集合: " + array);
iterator.remove();
System.out.println("将获得的下一个元素删除后的集合: " + array);
}
}

```

运行结果如图 10.13 所示。



图 10.13 使用迭代器输入 ArrayList 中的元素

10.4.2 范例 5: Iterator 遍历 ArrayList

ArrayList 在以后的开发中会经常用到, 本范例将练习使用 Iterator 和 for 循环组合遍历集合。运行结果如图 10.14 所示。(实例位置: 光盘\TM\sl\10\13)



图 10.14 使用 Iterator 遍历 ArrayList

在项目中新建 IteratorDemo 类，在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素，利用迭代器遍历 ArrayList 集合，其循环条件为如果迭代器中仍有元素可以迭代则继续循环，如果没有则跳出循环。代码如下：

```
public class IteratorDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();           //创建列表
        for (int i = 0; i < 10; i++) {                             //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的全部元素：");
        for (Iterator<Integer> it = list.iterator(); it.hasNext();) {
            System.out.print(it.next()+" ");
        }
    }
}
```

10.4.3 范例 6: ListIterator 遍历 ArrayList

对于列表而言，除了 Iterator，Java 语言还提供了一个功能更加强大的 ListIterator，它可以实现逆序遍历列表中的元素。本范例将使用其逆序遍历 ArrayList。运行结果如图 10.15 所示。（实例位置：光盘\TM\sl\10\14）

在项目中新建 ListIteratorDemo 类，在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素，获得迭代器对象后利用 hasPrevious()方法逆序输出 ArrayList 集合中的元素。代码如下：

```
public class ListIteratorDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();           //创建列表
        for (int i = 0; i < 10; i++) {                             //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的全部元素：" + list);
        System.out.println("逆序输出列表中的元素：");
    }
}
```



图 10.15 逆序遍历 ArrayList

```

ListIterator<Integer> li = list.listIterator();           //获得 ListIterator 对象
for (li = list.listIterator(); li.hasNext();){          //将游标定位到列表结尾
    li.next();
}
for (; li.hasPrevious();){                               //逆序输出列表中的元素
    System.out.print(li.previous() + " ");
}
}
}

```

10.5 经典范例

10.5.1 经典范例 1：制作电子词典

 视频讲解：光盘\TM\lx\10\制作电子词典.exe

词典通常用于解释一个词的含义，这是一种映射关系，因此可以使用 Map 来实现。本范例将制作一个电子词典。运行结果如图 10.16 所示。

（实例位置：光盘\TM\lx\10\15）

（1）在项目中新建窗体类 DictionaryDemo，在窗体中添加标签等控件。代码如下：



图 10.16 电子词典

```

public DictionaryDemo() {
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowActivated(WindowEvent e) {
            do_this_windowActivated(e);
        }
    });
    setTitle("\u6211\u7684\u7535\u5b50\u8bcd\u5178");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);

    JPanel panel1 = new JPanel();
    contentPane.add(panel1, BorderLayout.NORTH);

    JLabel label = new JLabel("\u8bf7\u8f93\u5165\u8981\u67e5\u8be2\u7684\u5355\u8bcd");
    panel1.add(label);

    JTextField textField = new JTextField();
    panel1.add(textField);
}

```

```

textField.setColumns(10);

JPanel panel2 = new JPanel();
contentPane.add(panel2, BorderLayout.SOUTH);

JButton button = new JButton("\u67E5\u8BE2");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        do_button_actionPerformed(e);
    }
});
panel2.add(button);

JScrollPane scrollPane = new JScrollPane();
contentPane.add(scrollPane, BorderLayout.CENTER);

textArea = new JTextArea();
textArea.setEditable(false);
scrollPane.setViewportView(textArea);
}

```

(2) 编写 `do_this_windowActivated()` 方法完成窗体激活事件监听, 在该方法中向 `Map` 中增加数据。代码如下:

```

protected void do_this_windowActivated(WindowEvent e) {
    words = new HashMap<String, String>();
    words.put("apple", "苹果");
    words.put("banana", "香蕉");
    words.put("water", "水");
}

```

(3) 编写 `do_button_actionPerformed()` 方法, 用来响应按钮单击事件。在该方法中完成了对单词的查询操作, 并根据不同的情况进行了提示。代码如下:

```

protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText(); //获得用户输入的单词
    if (text.isEmpty()) { //如果用户输入为空则提示用户
        JOptionPane.showMessageDialog(this, "请输入要查询的单词!", null,
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String meaning = words.get(text); //查询单词的含义
    if (meaning == null) { //如果没有这个单词则提示用户
        JOptionPane.showMessageDialog(this, "要查询的单词不存在!", null,
            JOptionPane.WARNING_MESSAGE);
        return;
    } else { //显示单词的含义
        textArea.setText(meaning);
    }
}

```

10.5.2 经典范例 2：制作手机电话本

 视频讲解：光盘\TMlx\10\制作手机电话本.exe

为了便于保存联系方式，手机都提供电话簿功能。它也是一种映射关系，因此可以使用 Map 来实现。本范例将制作一个手机电话簿。运行结果如图 10.17 所示。（实例位置：光盘\TMsl\10\16）

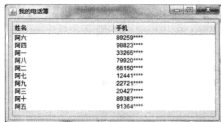


图 10.17 手机电话簿

（1）在项目中新建窗体类 Phonebook，在窗体中添加表格控件。代码如下：

```
public Phonebook() {
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowActivated(WindowEvent e) {
            do_this_windowActivated(e);
        }
    });
    setTitle("\u6211\u7684\u7535\u8bdd\u66f4\u6539");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 250);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);

    JScrollPane scrollPane = new JScrollPane();
    contentPane.add(scrollPane, BorderLayout.CENTER);

    table = new JTable();
    scrollPane.setViewportView(table);
}
```

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听，在该方法中向表格模型中增加数据，最后更新表格模型。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    Map<String, String> directory = new HashMap<String, String>(); //创建集合
    directory.put("阿一", "33265****"); //向集合中增加元素
}
```

```

directory.put("阿二", "66150*****");           //向集合中增加元素
directory.put("阿三", "20427*****");           //向集合中增加元素
directory.put("阿四", "98823*****");           //向集合中增加元素
directory.put("阿五", "91364*****");           //向集合中增加元素
directory.put("阿六", "89259*****");           //向集合中增加元素
directory.put("阿七", "12441*****");           //向集合中增加元素
directory.put("阿八", "79920*****");           //向集合中增加元素
directory.put("阿九", "22721*****");           //向集合中增加元素
directory.put("阿十", "89383*****");           //向集合中增加元素
DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得表格模型
model.setColumnIdentifiers(new Object[] { "姓名", "手机" }); //设置表头
Set<String> names = directory.keySet();          //获得键集合
for (Iterator<String> it = names.iterator(); it.hasNext();) {
    String name = it.next();                     //获得键
    model.addRow(new Object[] { name, directory.get(name) }); //向表格中增加元素
}
table.setModel(model);                          //更新表格模型
}

```

10.6 本章小结

本章向读者介绍了 Java 语言中常见的集合，包括 List 集合、Set 集合、Map 集合。对于每种集合的特点读者应该有所了解，重点掌握集合的遍历、添加对象、删除对象的方法。本章在介绍每种集合时都给出了典型实用的小例子，以帮助读者掌握集合类的常用方法。集合是 Java 语言中很重要的部分，通过本章的学习，读者应该学会使用集合类。

10.7 实战练习

1. 将 1~100 之间的所有正整数存放在一个 List 集合中，并将集合中索引位置是 10 的对象从集合中移除。（答案位置：光盘\TM\10\17）
2. 分别向 Set 集合以及 List 集合中添加“A”、“a”、“c”、“C”、“a”5 个元素，观察重复值“a”能否在 List 集合以及 Set 集合中成功添加。（答案位置：光盘\TM\10\18）
3. 创建 Map 集合，创建 Emp 对象，并将创建的 Emp 对象添加到集合中（Emp 对象的 id 作为 Map 集合的键），并将 id 为 005 的对象从集合中移除。（答案位置：光盘\TM\10\19）

