

Java 应用程序设计

集合与映射

王晓东

wxd2870@163.com

中国海洋大学

November 2, 2017



参考书目

1. 张利国、刘伟 [编著], Java SE 应用程序设计, 北京理工大学出版社, 2007.10.



本章学习目标

1. 列表（List）
2. 集（Set）
3. 映射（Map）
4. 其它相关 API



大纲

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



集合框架概述

集合就是将若干用途、性质相同或相近的“数据”组合而成一个整体。从体系上讲，集合类型可以归纳为三种：

集 Set 集合中不区分元素的顺序，不允许出现重复元素。例如应用于记录所有用户名的场合。

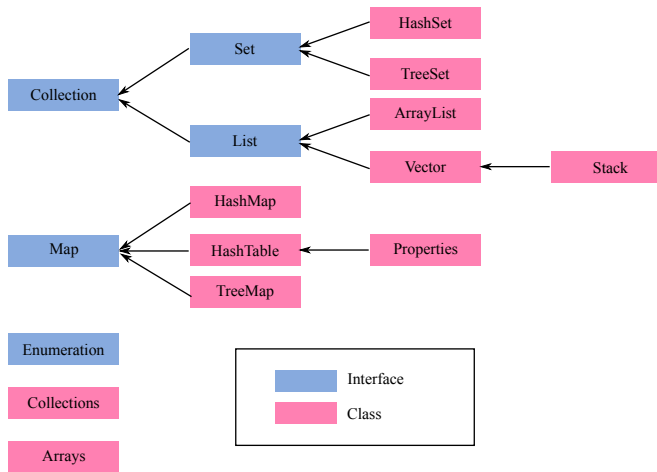
列表 List 集合区分元素的顺序，且允许包含重复元素。相当于数据结构中的线性表，具体表现为数组和向量、链表、栈、队列等。

映射 Map 中保存成对的“键 - 值”（Key - Value）信息，映射中不能包含重复的键，每个键最多只能映射一个值。

Java 集合中只能保存引用类型的数据，实际上存放的是对象的引用而非对象本身。Java API 中的集合类型均定义在 `java.util` 包中。



集合相关 API 关系结构



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ **boolean add(Object o)**
向集合中添加一个元素，在子接口中此方法发生了分化，如 Set 接口中添加重复元素时会被拒绝（返回 false，而不是出错）；List 接口则会接受重复元素且返回 true。
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
从集合中移除指定的元素。
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
返回集合中元素的数目。
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ **boolean isEmpty()**
判断集合是否为空（即是否包含任何元素）。
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
判断集合中是否包含指定的元素。
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
移除当前集合中的所有元素。
- ▶ Iterator iterator()
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ **Iterator iterator()**
返回在此集合的元素上进行迭代的迭代器。
- ▶ Object[] toArray()



Collection 接口

java.util.Collection 接口是描述 Set 和 List 集合类型（不包含 Map）的根接口，其中定义了有关集合操作的普遍性方法：

- ▶ boolean add(Object o)
- ▶ boolean remove(Object o)
- ▶ int size()
- ▶ boolean isEmpty()
- ▶ boolean contains(Object o)
- ▶ void clear()
- ▶ Iterator iterator()
- ▶ **Object[] toArray()**
返回包含当前集合中所有元素的数组。



Set 和 List 接口

java.util.Set 和 java.util.List 分别描述前述的集和列表结构，二者均为 Collection 的子接口。

Set 接口模拟了数学意义的集合。

List 接口规定使用者可以对列表元素的插入位置进行精确控制，并添加了根据元素索引来访问元素等功能，接口中新添加了相应方法：

- ▶ **void add(int index, Object element)**
- ▶ Object get(int index)
- ▶ Object set(int index, Object element)
- ▶ int indexOf(Object o) 返回列表中首次出现指定元素的索引，如果列表不包含指定元素，则返回 -1。
- ▶ Object remove(int index)



Set 和 List 接口

java.util.Set 和 java.util.List 分别描述前述的集和列表结构，二者均为 Collection 的子接口。

Set 接口模拟了数学意义的集合。

List 接口规定使用者可以对列表元素的插入位置进行精确控制，并添加了根据元素索引来访问元素等功能，接口中新添加了相应方法：

- ▶ void add(int index, Object element)
- ▶ **Object get(int index)**
- ▶ Object set(int index, Object element)
- ▶ int indexOf(Object o) 返回列表中首次出现指定元素的索引，如果列表不包含指定元素，则返回 -1。
- ▶ Object remove(int index)



Set 和 List 接口

java.util.Set 和 java.util.List 分别描述前述的集和列表结构，二者均为 Collection 的子接口。

Set 接口模拟了数学意义的集合。

List 接口规定使用者可以对列表元素的插入位置进行精确控制，并添加了根据元素索引来访问元素等功能，接口中新添加了相应方法：

- ▶ void add(int index, Object element)
- ▶ Object get(int index)
- ▶ **Object set(int index, Object element)**
- ▶ int indexOf(Object o) 返回列表中首次出现指定元素的索引，如果列表不包含指定元素，则返回 -1。
- ▶ Object remove(int index)



Set 和 List 接口

java.util.Set 和 java.util.List 分别描述前述的集和列表结构，二者均为 Collection 的子接口。

Set 接口模拟了数学意义的集合。

List 接口规定使用者可以对列表元素的插入位置进行精确控制，并添加了根据元素索引来访问元素等功能，接口中新添加了相应方法：

- ▶ void add(int index, Object element)
- ▶ Object get(int index)
- ▶ Object set(int index, Object element)
- ▶ int indexOf(Object o) 返回列表中首次出现指定元素的索引，如果列表不包含指定元素，则返回 -1。
- ▶ Object remove(int index)



Set 和 List 接口

java.util.Set 和 java.util.List 分别描述前述的集和列表结构，二者均为 Collection 的子接口。

Set 接口模拟了数学意义的集合。

List 接口规定使用者可以对列表元素的插入位置进行精确控制，并添加了根据元素索引来访问元素等功能，接口中新添加了相应方法：

- ▶ void add(int index, Object element)
- ▶ Object get(int index)
- ▶ Object set(int index, Object element)
- ▶ int indexOf(Object o) 返回列表中首次出现指定元素的索引，如果列表不包含指定元素，则返回 -1。
- ▶ Object remove(int index)



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ **Object put(Object key, Object value)**
向当前映射中加入一组新的键—值对，并返回所加入元素的“值”，如果此映射中以前包含一个该键的映射关系，则用新值替换旧值。
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
返回此映射中映射到指定键的值，没有则返回 null。
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ **boolean isEmpty()**
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ **void clear()**
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ **int size()**
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ **boolean containsKey(Object key)**
如果映射中包含指定键的映射关系，则返回 true，否则返回 false。
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键-值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ **boolean containsValue(Object value)**
- ▶ Set keySet()
- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()

返回此映射中包含的键的 set 视图，此 Set 受映射支持，所以对映射的改变可以在此 Set 中反映出来，反之亦然。

- ▶ Collection values()



Map 接口

java.util.Map 接口描述了映射结构，Map 结构允许以键集、值集合或键—值映射关系集的形式查看某个映射的内容。主要方法：

- ▶ Object put(Object key, Object value)
- ▶ Object get(Object key)
- ▶ boolean isEmpty()
- ▶ void clear()
- ▶ int size()
- ▶ boolean containsKey(Object key)
- ▶ boolean containsValue(Object value)
- ▶ Set keySet()
- ▶ Collection values()
返回此映射包含值的 Collection 视图，此 Collection 受映射支持，所以对映射的改变可以在此 Collection 中反映出来，反之亦然。



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



ArrayList 类

java.util.ArrayList 类实现了 List 接口，用于表述长度可变的数组列表。

ArrayList 列表允许元素取值为 null。除实现了 List 接口定义的所有功能外，还提供了一些方法来操作列表容量的大小，相关方法包括：

- ▶ public ArrayList()
构造方法：创建一个初始容量为 10 的空列表。
- ▶ public ArrayList(int initialCapacity)
- ▶ public void ensureCapacity(int minCapacity)
- ▶ public void trimToSize()
将此 ArrayList 实例的容量调整为列表的当前大小。



Vector 类

java.util.Vector 也实现了 List 接口，其描述的也是可变长度的对象数组。

❖ 与 ArrayList 的差别

Vector 是同步（线程安全）的，运行效率要低一些，主要用在在多线程环境中，而 ArrayList 是不同步的，适合在单线程环境中使用。

常用方法（除实现 List 接口中定义的方法外）：

- ▶ public Vector()
- ▶ public Object elementAt(int index)
- ▶ public void addElement(Object obj)
- ▶ public void removeElementAt(int index)
- ▶ public void insertElementAt(E obj, int index)
- ▶ public boolean removeElement(Object obj)
- ▶ public void removeAllElements()
- ▶ public Object[] toArray()



Stack

java.util.Stack 类继承了 Vector 类，对应数据结构中以“后进先出”（Last in First out, LIFO）方式存储和操作数据的对象栈。

Stack 类提供了常规的栈操作方法：

- ▶ public Stack() 构造方法，创建一个空栈。
- ▶ public Object push(E item) 向栈中压入数据。
- ▶ public Object pop() 移除栈顶对象并作为此方法的返回值。
- ▶ public Object peek() 查看/返回栈顶对象，但不从栈中移除它。
- ▶ public boolean empty() 测试栈是否为空。
- ▶ public void clear() 清空栈。
- ▶ public int search(Object o) 返回对象在栈中的位置，以 1 为基数。



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



Iterator 接口

对于 ArrayList 可以使用 get() 方法访问其元素，而对于 Vector，还可以使用 elementAt() 方法访问其元素，后续 Set 和 Map 集合也有各自不同的元素访问方式。

是否有一种统一的方式来遍历各种不同类型集合中的元素呢？



Iterator 接口

对于 ArrayList 可以使用 get() 方法访问其元素，而对于 Vector，还可以使用 elementAt() 方法访问其元素，后续 Set 和 Map 集合也有各自不同的元素访问方式。

是否有一种统一的方式来遍历各种不同类型集合中的元素呢？



Iterator 接口

Java.util.Iterator 接口描述的是以统一方式对各种集合元素进行遍历/迭代的工具，也称为“迭代器”。

迭代器允许在遍历过程中移除集合中的（当前遍历到的那个）元素。主要方法包括：

- ▶ boolean hasNext()
如果仍有元素可以迭代，则返回 true，否则返回 false。
- ▶ Object next()
返回迭代的下一个元素，重复调用此方法直到 hasNext() 方法返回 false。
- ▶ void remove()
将当前迭代到的元素从迭代器指向的集合中移除。



使用迭代器

我们一般不直接创建迭代器对象，而是通过调用集合对象的 `iterator()` 方法（该方法在 `Collection` 接口中定义）来获取。

CODE TestIterator.java

```
1  import java.util.ArrayList;
2  import java.util.Iterator;

4  public class TestIterator {
5      public static void main(String[] args) {
6          ArrayList a = new ArrayList();
7          a.add("China");
8          a.add("USA");
9          a.add("Korea");
10         Iterator it = a.iterator();

12         while(it.hasNext()) {
13             String country = (String) it.next();
14             System.out.println(country);
15         }
16     }
17 }
```

注意：迭代器相当于原始集合的一个“视图”，即一种表现形式，而不是复制其中所有元素得到的拷贝，因此在迭代器上的操作将影响到原来的集合。



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



HashSet 类

`java.util.HashSet` 类实现了 `java.util.Set` 接口，描述典型的 Set 集合结构。

- ▶ `HashSet` 中不允许出现重复元素，不保证集合中元素的顺序。
- ▶ `HashSet` 中允许包含值为 `null` 的元素，但最多只能有一个 `null` 元素。



TreeSet 类

`java.util.TreeSet` 类也实现了 `java.util.Set`，它描述的是 `Set` 的一种变体——可以实现排序功能的集合。

在将对象元素添加到 `TreeSet` 集中时会自动按照某种比较规则将其插入到有序的对象序列中，以保证 `TreeSet` 集合元素组成的对象序列时刻按照“升序”排列（后续介绍升序，例如按照字典顺序排列），对于用户自定义类型的数据我们可以自行定义其所需的排序规则。



Comparable 接口

java.lang.Comparable 接口中定义的 compareTo() 方法用于提供对其实现类的对象进行整体排序所需的比较逻辑，所为的排序可以理解为按照某种标准来比较对象的大小，以确定其次序。

- ▶ 实现类基于 compareTo() 方法的排序被称为自然排序。而 compareTo() 方法被称为它的自然比较方法，具体的排序原则可由实现类根据需要而定。方法格式如下：

```
int compareTo(Object o)
```



Comparable 接口

❖ 使用 Comparable 接口实现自然排序

CODE ♦ Person.java

```
1 public class Person implements java.lang.Comparable {
2     private final int id;
3     ...
4
5     public Person(int id, String name, int age) {
6         this.id = id;
7         ...
8     }
9     ...
10    @Override
11    public int compareTo(Object o) {
12        Person p = (Person) o;
13        return this.id - p.id;
14    }
15    @Override
16    public boolean equals(Object o) {
17        boolean flag = false;
18        if (o instanceof Person) {
19            if (this.id == ((Person) o).id) {
20                flag = true;
21            }
22        }
23        return flag;
24    }
25 }
```



Comparable 接口

CODE TestComparable.java

```
1  import java.util.TreeSet;
2  import java.util.Iterator;

4  public class TestComparable {
5      public void static main(String[] args) {
6          TreeSet ts = new TreeSet();
7          ts.add(new Person(1003, "Bob", 15));
8          ts.add(new Person(1008, "Alice", 25));
9          ts.add(new Person(1001, "Kevin", 30));
10     }
11     Iterator it = ts.getIterator();
12     while (it.hasNext()) {
13         Person employee = (Person) it.next();
14         System.out.println(employee);
15     }
16 }
```

output

Id: 1001 Name: Kevin Age:30

Id: 1003 Name: Bob Age:15

Id: 1008 Name: Alice Age:25



Comparable 接口

❖ 对上述程序的几点说明

1. 用户在重写 `compareTo()` 方法以定制比较逻辑时，需要确保其与等价性判断方法 `equals()` 保持一致，即确保条件“(x.compareTo(y) == 0) == (x.equals(y))”永远成立，否则逻辑上不合理。所以上例同时重写了 `equals()` 方法。
2. 为保证能够实现元素的排序功能，`TreeSet` 集合要求向其加入的对象元素必须是 `Comparable` 接口的实现类的实例，否则程序运行时会抛出造型异常 (`java.lang.ClassCastException`)。
3. `Comparable` 接口并不专用于集合框架。



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



HashMap 类

java.util.HashMap 类实现了 java.util.Map 接口，该类基于哈希表实现了前述的映射集合结构。

- ▶ HashMap 结构不保证其中元素（映射信息）的先后顺序，且允许使用 null “值” 和 null “键”。
- ▶ 当集合中不存在当前检索的“键”所对应的映射值时，HashMap 的 get() 方法会返回空值 null，而不会运行出错。
- ▶ 影响 HashMap 性能的两个参数：初始容量（Initial Capacity）和加载因子（Load Factor）。



HashTable 类

java.util.Hashtable 与 HashMap 作用基本相同，也实现了 Map 接口，采用哈希表的方式将“键”映射到相应的“值”。

❖ Hashtable 与 HashMap 的差别

- ▶ Hashtable 中元素的“键”和“值”均不允许为 null，而 HashMap 则允许。
- ▶ Hashtable 是同步的，即线程安全的，效率相对要低一些，适合在多线程环境下使用；而 HashMap 是不同步的，效率相对高一些，提倡在单线程环境中使用。
- ▶ 除此之外，Hashtable 与 HashMap 的用法格式完全相同。



TreeMap 类

java.util.TreeMap 类实现了将 Map 映射中的元素按照“键”进行升序排列的功能，其排序规则可以是默认的按照“键”的自然顺序排列，也可以使用指定的其他排序规则。

向 TreeMap 映射中添加的元素“键”所属的类必须实现 Comparable 接口。

```
1 public MyKey implements Comparable {
2     private final int id;
3     ...
4     public MyKey(int id) {
5         this.id = id;
6     }
7     ...
8     @Override
9     public int compareTo(Object o) {
10         return this.id - ((MyKey) o).id;
11     }
12     @Override
13     public boolean equals(Object o) {
14         return (o instanceof MyKey) && (this.id == ((MyKey) o).id);
15     }
16     @Override
17     public int hashCode() {
18         return new Integer(id).hashCode();
19     }
20 }
```



TreeMap 类

❖ 对上述程序的说明

MyKey 类重写 equals() 方法的同时也重写了 hashCode() 方法，这是一种规范的做法，目的是为了维护 hashCode() 方法的常规协定，该协定要求相等对象必须具有相等的哈希码，即当两个对象使用 equals() 方法比较结果为等价时，它们各自调用 hashCode() 方法也应该返回相同的结果。



接下来...

[集合框架概述](#)

[Collection 及 Map 接口](#)

[列表](#)

[Iterator 接口](#)

[集](#)

[映射](#)

[其他相关 API](#)



Enumeration 接口

`java.util.Enumeration` 接口作用与 `Iterator` 接口类似，但只提供了遍历 `Vector` 和 `Hashtable`（及子类 `Properties`）类型集合元素的功能，且不支持集合元素的移除操作。

```
1      import java.util.*;
3
3      public class TestEnumeration {
4          public static void main(String[] args) {
5              Vector v = new Vector();
6              v.addElement("Lisa");
7              v.addElement("Billy");
8              v.addElement("Brown");
10
10             Enumeration e = v.elements();
12
12             while(e.hasMoreElements()) {
13                 String value = (String) e.nextElement();
14                 System.out.println(value);
15             }
16         }
17     }
```



Collections 类

`java.util.Collections` 类定义了多种集合操作方法，能够实现了对集合元素的排序、取极值、批量拷贝、集合结构转换、循环移位以及匹配性检查等功能。Collections 类的主要方法包括：

- ▶ `public static void sort(List list)`
- ▶ `public static void reverse(List list)`
- ▶ `public static void shuffle(List list)`
- ▶ `public static void copy(List dest, List src)`
- ▶ `public static ArrayList list(Enumeration e)`
- ▶ `public static int frequency(Collection c, Object o)`
- ▶ `public static T max(Collection coll)`
- ▶ `public static T min(Collection coll)`
- ▶ `public static void rotate(List list, int distance)`



Arrays 类

`java.util.Arrays` 类定义了多种数组操作方法，实现了对数组元素的排序、填充、转换为列表或字符串形式、增强的检索和深度比较等功能。Arrays 类的主要方法包括¹：

- ▶ `public static List asList(Object... a)`
- ▶ `public static void sort(< 类型 >[] a)`
- ▶ `public static int binarySearch(int[] a, int key)`
- ▶ `public static String toString(int[] a)`

¹自行搜索学习各方法的使用法



THE END

wxd2870@163.com

