Collection與Map

學習目標

- 泛型
- 認識Collection與Map架構
- 使用Collection與Map實作物件
- 對收集物件進行排序

什麼是泛型

- 泛型的英文是Generics，是指在定義方法、介面或類的時候，不預先指定具體的型別，使用的時候再指定型別。
- 泛型（Generics）是在JDK1.5中推出的，其主要目的是可以建立具有類型安全的集合架構，如列表、映射等資料結構。

泛型類別的定義

為了和普通的類有所區別，這樣宣告的類別稱作泛型類，如：

```
class A< E >          class A< E,K >{
                                    }
```

- A是泛型類別的名稱，E是泛型，E沒有指定是何種類別的資料，它可以是任何物件或或介面，但不能是基本類別資料。
- 類型參數宣告部分包含一個或多個類別參數，參數間用逗號隔開。
- 泛型參數也被稱為泛型類別變數，用於指定泛型類別名稱的識別字。

- 泛型類別參數：

T：代表這是一個類別Type

E：代表這是一個元素Element

K：代表這是一個鍵Key

V：代表這是一個值Value

？:表示不确定的 java 类型

單個型別引數的泛型類別定義

```
public class Info<T> {
    private T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
}
```

多個型別引數的泛型類別定義

```
public class Pair<T, K> {
    private T first;
    private K last;

    public Pair(T first, K last) {
        this.first = first;
        this.last = last;
    }
    public T getFirst() {
        return first;
    }
    public K getLast() {
        return last;
    }
}
```

泛型方法的定義

1. 泛型方法

方法也可以是泛型方法，泛型方法可以定義在泛型類中，也可以定義在非泛型類中。泛型方法定義形式：

```
[訪問限定詞] [static]<類型參數表列> 方法類型 方法名([參數表列])
{
    //…
}
```

泛型方法的定義

```
package ch9_1;
public class ch9_1 {
    public static <E> void printArray(E[] arr) {
        for (E ele : arr) {
            System.out.print(ele + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        // 建立整數, 浮點數和字元陣列
        Integer[] intArr = { 1, 2, 3, 4, 5 };
        Double[] doubleArr = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArr = { 'H', 'E', 'L', 'L', 'O' };
        // 顯示陣列元素
        System.out.print("整數: ");
        printArray(intArr); // 整數
        System.out.print("浮點數: ");
        printArray(doubleArr); // 浮點數
        System.out.print("字元: ");
        printArray(charArr); // 字元
    }
}
```

結果:

```
整數: 1 2 3 4 5
浮點數: 1.1 2.2 3.3 4.4
字元: H E L L O
```

泛型方法的定義

2. 具有可變參數的方法

[訪問限定詞] <類型參數表列> 方法類型 方法名(類型參數名… 參數名)

{

//……

}

利用泛型方法，可以定義具有可變參數的方法，如printf方法：

System.out.printf(“%d,%f\n” ,i,f);

System.out.printf(“x=%d,y=%d,z=%d” ,x,y,z);

printf是具有可變參數的方法。

泛型參數的限定

- 如果只接收指定範圍內的類別，可以對泛型的參數進行限定。

參數限定的語法形式：

類別形式參數 extends 父類

- “類別形式參數”是指宣告泛型類別時所宣告的類別，“父類”表示只有這個類別下面的子類別才可以做實際類別繼承。

泛型類別物件的定義

定義泛型類別的物件：

泛型類名[<實際類型表列>] 物件名=new 泛型類名[<實際類型表列>]([形參表]);

或

泛型類名[<實際類型表列>] 物件名=new 泛型類名[<>]([形參表]);

也可以用“?”代替“實際參數表列”

如：

```
Basket<E> basket = new Basket<E>();
```

泛型類別的定義

```
package ch9_2;
public class Box<T> {// Box類別宣告
    private T t;
    public void set(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
}
```

```
package ch9_2;
public class ch9_2 {
    public static void main(String[] args) {
      // 分別建立整數和字串的Box物件
      Box<Integer> intBox = new Box<Integer>();
      Box<String> strBox = new Box<String>();
      // 指定值
      intBox.set(5);
      strBox.set("Hello World");
      // 顯示值
      System.out.println("整數: " + intBox.get());
      System.out.println("字串: " + strBox.get());
    }
}
```

結果:

```
整數: 5
字串: Hello World
```

泛型介面的定義

泛型介面定義形式：

```
interface 介面名<類型參數表列>
{
//……
}
```

```
public interface Content<T> {
    T text();
}
```

泛型介面的實現

實現介面形式如下：

```
class 類名<類型參數表列> implements 介面名<類型參數表列>
{
//…
}
```

泛型介面的實現

1.子類沒有泛型型別

```
public class IntContent implements Content<Integer> {
	private int text;

	public IntContent(int text) {
		this.text = text;
	}

	public Integer text() {
		return text;
	}
}
```

泛型介面的實現

2.子類不明確宣告泛型型別

```
public class GenericsContent<T> implements Content<T> {
    private T text;

    public GenericsContent(T text) {
        this.text = text;
    }

    public T text() {
        return text;
    }
}
```

泛型介面的實現2-1

```
package ch9_3;
public interface showi<T> {
    public void show(T t);
}

package ch9_3;
public class showc1 implements showi<String> {
    public void show(String t) {
        System.out.println("show:" + t);
    }
}

package ch9_3;
public class showc2<T> implements showi<T> {
    public void show(T t) {
        System.out.println("show:" + t);
    }
}
```

泛型介面的實現2-2

```
package ch9_3;
public class ch9_3 {
    public static void main(String[] args) {
        showc2<Integer> obj1 = new showc2<Integer>();
        obj1.show(6);
        showc1 obj2 = new showc1();
        obj2.show("java");
     }
}
```

結果: show:6
show:java

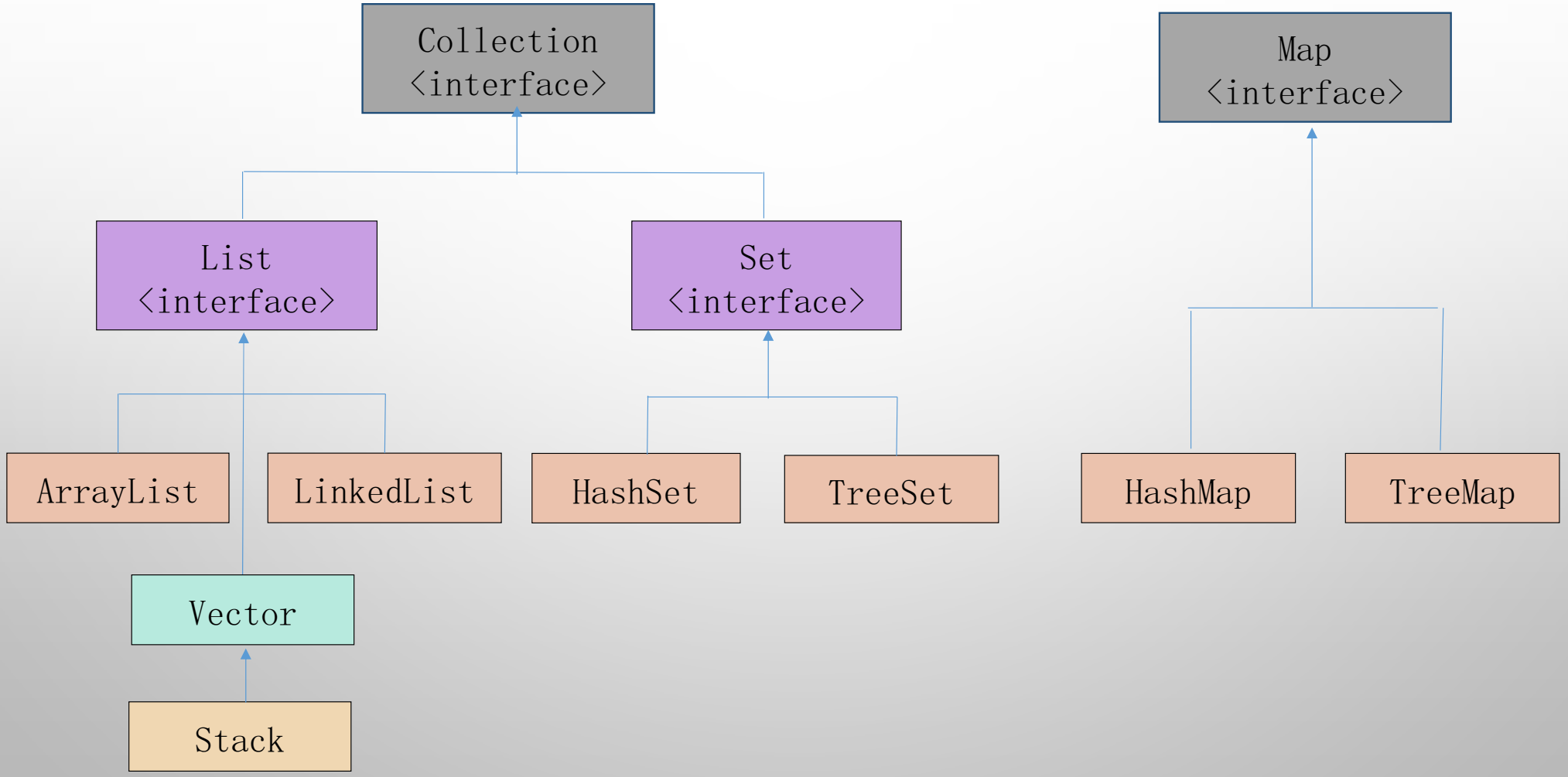
Java的資料結構

Java資料結構中傳統的介面和類別：

- 列舉（Enumeration）
- 位元集合（BitSet）
- 向量（Vector）
- 堆疊（Stack）
- 字典（Dictionary）
- 雜湊表（Hashtable）
- 屬性（Properties）

在Java2中引入了一種新的架構-Java Collections Framework (JCF) Java收集架構

Java的收集類別


Collection
<interface>
Map
<interface>
List
<interface>
Set
<interface>
ArrayList
LinkedList
HashSet
TreeSet
HashMap
TreeMap
Vector
Stack

Java的收集類別

- Collection 是最基本的收集介面，一個 Collection 代表一組 Object，即 Collection 的元素,Java不提供直接繼承自Collection的類別，只提供繼承於的子介面(如List和set)。
- Collection介面的部分方法。

返回類型	方法名	方法功能
boolean	add(E e)	向集合中添加新元素
boolean	addAll(Collection c)	將指定集合中的所有元素添加到當前集合中。
boolean	remove(Object o)	刪除當前集合中包含的指定元素
boolean	removeAll(Collection c)	刪除當前集合中與指定集合相同的所有元素
boolean	retainAll(Collection c)	保留當前集合中與指定集合相同的所有元素
void	clear()	刪除當前集合中的所有元素
boolean	contains(Object o)	查找當前集合中是否有指定元素
boolean	containsAll(Collection c)	查找當前集合中是否包含指定集合中的所有元素
boolean	isEmpty()	當前集合是否為空
int	size()	返回當前集合的元素個數
Iterator	iterator()	返回一個可遍歷當前集合的反覆運算器
Stream	stream()	返回一個連接集合的順序流
Object[]	toArray()	返回一個當前集合所有元素的陣列

收集類別及其特點

收集類別的特點：

- 空間自主調整，提高空間利用率。
- 提供不同的資料結構和演算法，減少程式設計工作量。
- 提高程式的處理速度和品質。

注意：

- 收集類別不支援單一資料型別的存放和處理。如果確實需要存儲簡單類型資料可以先進行類的封裝（裝箱）處理。
- 收集類別中存放的是物件的呼叫，而不是物件本身

Java的收集類別

- Java中實現Collection收集介面的類，主要包括List列表介面和Set集合介面。
- List列表是一個有序元素，允許出現重複元素。
- Set集合是一個無序，不允許出現重複元素。
- Map映射是Java.util包中的另一個介面，Map中的每個元素都由一個鍵-值對構成。在Map中不能有重複的鍵，但是可以有相同的值。
- 收集類均採用泛型進行定義。

具有索引的List列表

- List是一種Collection，作用是收集物件，並以索引方式保留收集的物件順序。
- List 介面存儲一組不唯一，有序（插入順序）的物件。
- List介面允許有相同的元素。

List列表介面

實現List列表介面的常用類別有

- ArrayList
- LinkedList
- Vector
- Stack。

List列表介面

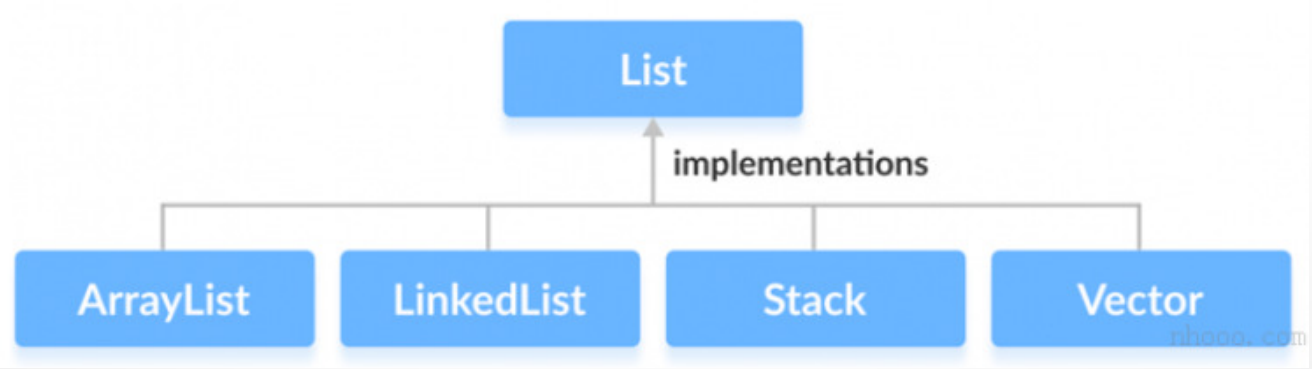
- List介面的定義形式：

  public interface List<E> extends Collection<E>

List介面中的主要方法

類型	方法名	方法功能
boolean	add(E e)	把元素e加到表的尾部
void	add(int index, E e)	把元素e加到表的index位置， 原index位置元素順序後移
boolean	equals(Object o)	比較物件o是否與表中的元素是 同一元素
E	get(int index)	得到表中index位置的元素
boolean	indexOf(Object o)	判斷元素o在表中是否存在。 如果不存在，則返回-1
Iterator<E>	iterator()	獲得表的遍歷器

如何使用List列表



導入 java.util.List 套件才能使用List。

1. 使用ArrayList 實現List

```
List<String> list1 = new ArrayList<>();
```

2.使用LinkedList 實現List

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

ArrayList 實現List

```
package ch9_4;
import java.util.List;
import java.util.ArrayList;
public class ch9_4 {
    public static void main(String[] args) {
        //使用ArrayList類創建列表
        List<Integer> numbers = new ArrayList<>();
        numbers.add(1); //將元素添加到清單
        numbers.add(2);
        numbers.add(3);
        System.out.println("List: " + numbers);
        int number = numbers.get(2); //從清單中訪問元素
        System.out.println("訪問元素: " + number);
        int removedNumber = numbers.remove(1); //從清單中刪除元素
        System.out.println("刪除元素: " + removedNumber);
    }
}
```

結果:

```
List: [1, 2, 3]
訪問元素: 3
刪除元素: 2
```

LinkedList 實現List

```
package ch9_5;
import java.util.List;
import java.util.LinkedList;
public class ch9_5 {
    public static void main(String[] args) {
        // 使用LinkedList類創建列表
        List<Integer> numbers = new LinkedList<>();
        numbers.add(1); // 將元素添加到清單
        numbers.add(2);
        numbers.add(3);
        System.out.println("List: " + numbers);
        int number = numbers.get(2); // 從清單中訪問元素
        System.out.println("訪問元素: " + number);
        int index = numbers.indexOf(2);// 使用indexOf()方法
        System.out.println("位置3的元素是 " + index);
        int removedNumber = numbers.remove(1);// 從清單中刪除元素
        System.out.println("刪除元素: " + removedNumber);
    }
}
```

結果:

```
List: [1, 2, 3]
訪問元素: 3
位置3的元素是 1
刪除元素: 2
```

ArrayList收集類

- ArrayList可看成是動態的陣列，可動態調整50%空間
- ArrayList利用索引值(index)在指定的位置進行修改。
- 資料儲存在連續的記憶體空間。

ArrayList類的定義：

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

ArrayList收集類

ArrayList類別的構造方法

方法名	方法功能
ArrayList()	建立一個初始容量為 10 的空列表。
ArrayList(Collection c)	建立一個包含指定 collection 的元素的清單。
ArrayList(int capacity)	建立一個具有指定初始容量的空清單。

ArrayList收集類

```
package ch9_6;
import java.util.ArrayList;
public class ch9_6 {
    public static void main(String[] args) {
        ArrayList<String> sites = new  ArrayList<String>();
        sites.add("Google");
        sites.add("Runoob");
        sites.add("Taobao");
        System.out.println(sites);
        System.out.println(sites.get(1)); // 取第2個元素
        sites.remove(2); // 刪除第3個元素
        System.out.println(sites);
    }
}
```

結果:

```
[Google, Runoob, Taobao]
Runoob
[Google, Runoob]
```

LinkedList類別

- LinkedList類別是另一個常用的線性清單列表，採用的是雙向列表結構。
- LinkedList在加入或刪除元素是從物件的起始或結尾處。
- 資料在記憶體位置是不連續的，有助於插入與刪除效率。

LinkedList類別的定義形式：

```
public class LinkedList<E>extends AbstractSequentialList<E>
implements List<E>, Queque<E>, Cloneable, Serializable
```

LinkedList收集類別

- LinkedList實現了兩個介面：List和Queue介面，List介面中定義了線性表操作方法， Queue是具有「First-In-First-Out」的資料結構。
- LinkedList物件既可以表示線性序列表，也可以把它當做堆疊使用，還可以把它當做佇列使用。

LinkedList收集類別

LinkedList類別的部分方法

返回類型	方法名	方法功能
boolean	add(E e)	將元素e加到列表的末尾
void	add(int index, E e)	把元素e插入到列表index所指位置，原位置元素順序後移
void	addFirst(E e)	將元素e插入到列表的頭部
E	getFirst()	返回清單的頭部元素
int	indexOf(Object o)	返回元素o在列表中第1次出現的位置。如果無元素o，則返回-1
boolean	offerFirst(E e)	將元素e插入到列表的頭部
E	pollLast()	返回清單中尾部元素並且從表中刪除該元素。如果表空，則返回null
E	pop()	堆疊頂端元素輸出
void	push(E e)	元素e輸入堆疊
E	removeFirst()	從清單中刪除頭部元素並返回該元素

LinkedList收集類別

```
 package ch9_7;
import java.util.LinkedList;
public class ch9_7 {
    public static void main(String[] args) {
        LinkedList mylist = new LinkedList();
        mylist.add("你"); // LinkedList中的第一個節點
        mylist.add("好"); // LinkedList中的第二個節點
        int number = mylist.size(); // 獲取LinkedList的長度
        for (int i = 0; i < number; i++) {
            String temp = (String) mylist.get(i); // 必須強制轉換取出的資料
            System.out.println("第" + i + "節點中的資料:" + temp);
        }
    }
}
```

結果:

```
第0節點中的資料:你
第1節點中的資料:好
```

EnumSet列舉類別

- EnumSet 類別是設置實現與列舉類型的使用。
- EnumSet的元素必須來自指定的，在創建時設置一個列舉類型。
- EnumSet在內部表示為位向量。
- EnumSet是不同步的。如果多個執行緒同時訪問一個列舉同時設置，並且至少有一個執行緒修改的設置，它應該保持外部同步。

方法	描述
EnumSet. allOf	創建一個包含所有在指定元素類型的元素的列舉set。
EnumSet. noneOf()	此方法創建一個空的列舉類別具有指定元素類型。

Java Enumeration介面

- Enumeration介面中定義了一些方法，通過這些方法可以列舉（一次獲得一個）物件集合中的元素。
- 使用在諸如Vector和Properties這些傳統類所定義的方法中，除此之外，還用在一些API類，並且在應用程式中也廣泛被使用。

方法	描述
hasMoreElements( )	測試此列舉是否包含更多的元素。
nextElement( )	如果此列舉物件至少還有一個可提供的元素，則返回此列舉的下一個元素。

EnumSet列舉類別

```
package ch9_8;
public enum Weeks {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURADAY, SUNDAY
}
```

```
package ch9_8;
import static ch9_8.Weeks.MONDAY;
import static ch9_8.Weeks.THURSDAY;
import java.util.EnumSet;
public class ch9_8 {
    public static void main(String[] args) {
        // create a set which is empty
        EnumSet<Weeks> week = EnumSet.noneOf(Weeks.class);
        week.add(MONDAY);
        System.out.println("EnumSet中的元素：" + week);
        week.remove(MONDAY);
        System.out.println("EnumSet中的元素：" + week);
        week.addAll(EnumSet.complementOf(week));
        System.out.println("EnumSet中的元素：" + week);
        week.removeAll(EnumSet.range(MONDAY, THURSDAY));
        System.out.println("EnumSet中的元素：" + week);
    }
}
```

結果:

```
EnumSet中的元素：[MONDAY]
EnumSet中的元素：[]
EnumSet中的元素：[MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURADAY, SUNDAY]
EnumSet中的元素：[FRIDAY, SATURADAY, SUNDAY]
```

Vector 向量類別

- Vector 類實現了一個動態陣列。和ArrayList很相似，但是兩者是不同的：
- Vector 是同步訪問的。
- Vector 包含了許多傳統的方法，這些方法不屬於收集框架。
- Vector 主要用在事先不知道陣列的大小，或者只是需要一個可以改變大小的陣列的情況。

Vector 類別支援 4 種建構式方法

- 創建一個默認的向量，默認大小為 10：
  Vector()
- 創建指定大小的向量。
  Vector(int size)
- 創建指定大小的向量，並且增量用 incr 指定。增量表示向量每次增加的元素數目。
  Vector(int size,int incr)
- 創建一個包含收集 c 元素的向量：
  Vector(Collection c)

Vector 類別支援 4 種建構式方法

方法	描述
add(int index, Object element)	在此向量的指定位置插入指定的元素。
add(Object o)	將指定元素添加到此向量的末尾。
addAll(Collection c)	將指定 Collection 中的所有元素添加到此向量的末尾，按照指定 collection 的反覆運算器所返回的順序添加這些元素。
addAll(int index, Collection c)	在指定位置將指定 Collection 中的所有元素插入到此向量中。
nextElement( )	返回列舉中的下一個物件.
hasMoreElements( )	列舉中還有更多的元素來提取它必須返回true或false.

Vector 類別

在向量中元素的操作

```
package ch9_9;
import java.util.Enumeration;
import java.util.Vector;

public class ch9_9 {
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<Integer>();
        for (int i = 0; i < 3; i++) {
            vector.add(i);
            System.out.println("在向量中增加元素：" + i);
        }
        Enumeration<Integer> e = vector.elements();
        while (e.hasMoreElements()) {
            System.out.println("獲得向量中的元素：" + e.nextElement());
        }
    }
}
```

結果:

```
在向量中增加元素：0
在向量中增加元素：1
在向量中增加元素：2
獲得向量中的元素：0
獲得向量中的元素：1
獲得向量中的元素：2
```

Stack堆疊類別

Java Stack 類別

- 堆疊是Vector的一個子類別，實現後進先出的堆疊。
- 堆疊只定義了預設構造函數，用來創建一個空堆疊。
- 堆疊除了包括由Vector定義的所有方法，也定義了自己的方法。

Stack堆疊類別

方法	描述
empty()	測試堆疊是否為空。
peek( )	查看堆疊頂部的物件，但不從堆疊中移除它。
pop( )	移除堆疊頂部的物件，並作為此函數的值返回該物件。
push(Object element)	把物件推入堆疊頂部。
search(Object element)	返回物件在堆疊中的位置，以 1 為基數。

Stack堆疊類別2-1

```
package ch9_10;
import java.util.*;
public class ch9_10 {
    static void showpush(Stack<Integer> st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }

    static void showpop(Stack<Integer> st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
```

Stack堆疊類別2-2

```
    public static void main(String[] args) {
        Stack<Integer> st = new   Stack<Integer>();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpop(st);
        showpop(st);

        try {
              showpop(st);
        } catch (EmptyStackException e) {
           System.out.println("empty stack");
        }
    }
}
```

結果:

```
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack
```

Set集合介面

- 集合的特性無序性：集合中每個元素的地位都是相同的，元素之間是無序的。
- 互異性：集合中任何兩個元素都認為是不相同的，即每個元素只能出現一次。

Set介面的定義形式：

```
public interface Set<E> extends Collection<E>
```

常用的集合運算方法

集合運算	對應方法	實現功能
並集	addAll(Collection c)	得到兩個集合的並集，結果保存到當前集合中
交集	retainAll(Collection c)	得到兩個集合的交集，結果保存到當前集合中
差集	removeAll(Collection c)	得到兩個集合的差集，結果保存到當前集合中
超集合	containsAll(Collection c)	當前集合是否包含集合c的所有元素，是返回true

Set集合介面

```
package ch9_11;
import java.util.*;
public class ch9_11 {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        System.out.println(set.add("abc")); // true
        System.out.println(set.add("xyz")); // true
        System.out.println(set.add("xyz")); // false，添加失敗，因為元素已存在
        System.out.println(set.contains("xyz")); // true，元素存在
        System.out.println(set.contains("XYZ")); // false，元素不存在(大寫)
        System.out.println(set.remove("hello")); // false，刪除失敗，因為元素不存在
        System.out.println(set.size()); // 2，一共兩個元素
    }
}
```

結果:

```
true
true
false
true
false
false
2
```

HashSet集合類別

- 元素不可存儲重複 ，沒有索引的方法，是無序的集合，存儲元素和取出元素的順序有可能不一致，雜湊表結構(查詢的速度非常快)

HashSet的定義形式：

- public class HashSet<E>extends AbstractSet<E>
- implements Set<E>, Cloneable, Serializable
- 該類實現了Set介面，為使用者提供快速查找和添加元素功能。

HashSet集合類別

HashSet的主要方法

返回类型	方法名	方法功能
	HashSet()	構造一個新的空 set，初始容量是 16，載入因數是 0.75。
	HashSet(Collection c)	構造一個包含指定 collection 中的元素的新 set。
	HashSet(int Capacity)	構造一個新的空 set，指定初始容量和預設的載入因數（0.75）。
	HashSet(int Capacity, float loadFactor)	構造一個新的空 set，指定初始容量和載入因數。
boolean	add(E e)	如果set 中沒有元素e，則添加。
void	clear()	刪除set中所有元素。
Object	clone()	返回此 HashSet 實例的副本，但並不複製這些元素本身。
boolean	contains(Object o)	查詢set中是否包含指定元素o
boolean	isEmpty()	判斷set中是否沒有任何元素。
Iterator<E>	iterator()	返回set中元素進行反覆運算的反覆運算器。
boolean	remove(Object o)	刪除set中的指定元素o
int	size()	返回set中的元素的數量（set的容量）。

HashSet集合類別2-1

```
package ch9_12;
import java.util.*;
public class ch9_12 {
    public static void main(String[] args) {
        HashSet<String> hset = new HashSet<>(); // 集合物件HashSet宣告
        String name0 = "阿凡達";
        String name1 = "終局之戰";
        System.out.println("新增元素前是否是空的=" + hset.isEmpty());
        hset.add("無限之戰"); // 新增元素
        hset.add(name0);
        hset.add("鐵達尼號");
        System.out.println("新增後尺寸=" + hset.size());// 顯示尺寸和是否是空的
        System.out.println("是否是空的=" + hset.isEmpty());
        System.out.print("HashSet內容: "); // 顯示集合物件內容
        System.out.println(hset);
        // 是否擁有指定元素
        System.out.println("HashSet是否有[" + name0 + "]: " + hset.contains(name0));
        System.out.println("HashSet是否有[" + name1 + "]: " + hset.contains(name1));
        hset.remove(name0); // 刪除元素
        System.out.print("HashSet刪除[" + name0 + "]:");
        System.out.println(hset);
        hset.clear(); // 清除集合物件
    }
}
```

HashSet集合類別2-2

結果:

新增元素前是否是空的=true
新增後尺寸=3
是否是空的=false
HashSet內容: [鐵達尼號, 阿凡達, 無限之戰]
HashSet是否有[阿凡達]: true
HashSet是否有[終局之戰]: false
HashSet刪除[阿凡達]:[鐵達尼號, 無限之戰]

TreeSet集合類別

1.無序：放入集合中的順序和從集合中取出的順序不相同。
2.傳入的值不能重複：放入集合中的值不能重複，重複則覆蓋。
3.傳入的值不能為null
4.排序：從集合中取出時必須按照其自然順序排序。

TreeSet類的定義形式：

- public class TreeSet<E>extends AbstractSet<E>
- implements NavigableSet<E>, Cloneable, Serializable

TreeSet集合類別

TreeSet的構造方法

方法名	方法功能
TreeSet()	構造一個新的空TreeSet，根據元素的自然順序進行排序。
TreeSet(Collection c)	構造一個包含指定 Collection 元素的新 TreeSet，按照元素的自然順序進行排序。

TreeSet集合類別

TreeSet類的部分方法

返回類型	方法名	方法功能
Comparator	comparator()	返回對此 set 中的元素進行排序的比較器。
Iterator<E>	iterator()	返回元素按昇冪進行反覆運算的反覆運算器。
Iterator<E>	descendingIterator()	返回元素按降冪進行反覆運算的反覆運算器。
E	first()	返回第一個元素。
E	last()	返回最後一個元素。
E	ceiling(E e)	返回大於等於給定元素的最小元素；如果不存在，則返回 null。
E	floor(E e)	返回小於等於給定元素的最大元素；如果不存在，則返回 null。
E	lower(E e)	返回小於給定元素的最大元素；如果不存在，則返回 null。
E	higher(E e)	返回大於給定元素的最小元素；如果不存在，則返回 null。

TreeSet集合類別

```
package ch9_13;
import java.util.*;
public class ch9_13 {
    public static void main(String[] args) {
        Set<String> set = new TreeSet<String>();
        set.add("abc");
        set.add("xyz");
        set.add("rst");
        System.out.println(set);// 可以直接輸出
        Iterator itSet = set.iterator();// 也可以遍歷輸出
    while (itSet.hasNext())
      System.out.print(itSet.next() + "\t");
      System.out.println();
    }
}
```

結果:

```
[abc, rst, xyz]
abcrstxyz
```

MAP映射介面

- Map要以關鍵值(key)儲存，這個關鍵值會對應到指定的資料，即對應值(value)。
- Map介面是以Map<K,V>表示，其中K是關鍵值Key，V是對應值Value。

Map介面的常用方法

返回類型	方法名	方法功能
void	clear()	刪除集合中的所有元素
boolean	containsKey(Object key)	查詢集合中是否存在指定的key鍵
boolean	containsValue(Object value)	查詢集合中是否存在指定的value值
Set<Map.Entry<K,V>>	entrySet()	返回集合中包含的所有鍵值對的Set集合視圖。
boolean	equals(Object o)	比較指定的物件與此集合物件是否相等。
V	get(Object key)	返回指定鍵所映射的值；如果鍵不存在則返回 null。

MAP映射介面

返回類型	方法名	方法功能
boolean	isEmpty()	查詢集合是否為空。
Set<K>	keySet()	返回集合中包含的所有鍵的Set集合視圖。
V	put(K key, V value)	添加一個鍵值對，如果存在該鍵，則替換原有的值。
void	putAll(Map m)	將集合m中所有鍵值對複製到當前集合中。
V	remove(Object key)	刪除指定鍵可以的鍵值對。
int	size()	返回集合中的鍵值對個數。
Collection<V>	values()	返回集合中包含的所有值的集合視圖。

HashMap映射類別

- HashMap中建立鍵值對應之後，鍵是無序的。
- HashMap是實作Map介面的類別，其儲存的元素分為關鍵值Key與對應值Value 。

HashMap類別的定義形式：

- public class HashMap<K,V>extends AbstractMap<K,V>
- implements Map<K,V>, Cloneable, Serializable

HashMap映射類別

HashMap的構造方法

方法名	方法功能
HashMap()	構造一個具有預設初始容量 (16) 和默認載入因數 (0.75) 的空 HashMap。
HashMap(int Capacity)	構造一個帶指定初始容量和預設載入因數 (0.75) 的空 HashMap。
HashMap(int Cap, float f)	構造一個帶指定初始容量和載入因數的空 HashMap。
HashMap(Map m)	構造一個映射關係與指定 Map 相同的新 HashMap。

HashMap映射類別2-1

```
package ch9_14;
import java.util.*;
public class ch9_14 {
    public static void main(String[] args) {
      // 集合物件HashMap宣告
        HashMap<String,String> hmap =new HashMap<>();
        String key = "Tom", name = "卡麥隆";
        System.out.println("新增元素前是否是空的=" +hmap.isEmpty());
        hmap.put("Joe",name);  // 新增元素
        hmap.put("Jane","泰諾克");
        hmap.put(key,"憂鬱之島");  hmap.put("Hueyan",name);
        System.out.println("新增後尺寸=" + hmap.size()); // 顯示尺寸和是否是空的
        System.out.println("是否是空的="+hmap.isEmpty()); // 顯示集合物件內容
        System.out.println("Hashmap內容: " + hmap); // 是否擁有指定元素
        System.out.println("Hashmap是否有[" + key +"]: " + hmap.containsKey(key));
        System.out.println("Hashmap是否有[" + name +"]: " + hmap.containsValue(name));
        System.out.print("鍵值: " + key); // 取得指定的值
        System.out.println(" --> 值: " + hmap.get(key));
```

HashMap映射類別2-2

```
        Set<String> keys = hmap.keySet(); // 轉換成Set和Collection物件
        System.out.println("Keys內容: " + keys);
        Collection<String> values = hmap.values();
        System.out.println("Values內容: " + values);
        hmap.remove(key);  // 刪除元素
        System.out.println("Hashmap刪除["+key+"]:" + hmap);
        hmap.clear();      // 清除集合物件
    }
}
```

結果:

```
新增元素前是否是空的=true
新增後尺寸=4
是否是空的=false
Hashmap內容: {Joe=卡麥隆, Tom=憂鬱之島, Hueyan=卡麥隆, Jane=泰諾克}
Hashmap是否有[Tom]: true
Hashmap是否有[卡麥隆]: true
鍵值: Tom --> 值: 憂鬱之島
Keys內容: [Joe, Tom, Hueyan, Jane]
Values內容: [卡麥隆, 憂鬱之島, 卡麥隆, 泰諾克]
Hashmap刪除[Tom]:{Joe=卡麥隆, Hueyan=卡麥隆, Jane=泰諾克}
```

TreeMap類別

- 無序，不允許重複（無序指元素順序與添加順序不一致）
- TreeMap預設對鍵進行排序，所以鍵必須實現自然排序或固定排序

TreeMap類別的定義形式：

```
public class TreeMap<K,V>extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

TreeMap類別

TreeMap類的構造方法

方法名	方法功能
TreeMap()	構造一個新的、空的集合。
TreeMap(Comparator c)	構造一個新的、空的集合，集合的排序方式由給定的比較器決定。
TreeMap(Map m)	構造一個與給定m相同的自然排序的新集合。
TreeMap(SortedMap m)	構造一個與指定有序集合m相同的新的集合。

TreeMap類別

TreeMap類的常用方法

返回類型	方法名	方法功能
Map.Entry	ceilingEntry(K key)	返回一個不小於給定鍵的最小鍵值對；如不存在，則返回 null。
K	ceilingKey(K key)	返回一個大於等於給定鍵的最小鍵；如不存在，則返回 null。
NavigableSet	**put(K key,V value)**	返回與key的先前一個值，則返回null，如果有鍵的映射關係。
Map.Entry	firstEntry()	返回此集合中最小鍵的鍵值對；如果集合為空，則返回 null。
K	firstKey()	返回此集合中第一個（最小）鍵。
V	get(Object key)	返回指定鍵所映射的值，如不存在，則返回 null。
Set	keySet()	返回此集合中所有鍵的Set視圖。
Map.Entry	lastEntry()	返回此集合中的最大鍵的鍵值對；如果為空，則返回 null。
Map.Entry	pollFirstEntry()	移除並返回與此集合中的最小鍵的鍵-值對；如果為空，則返回 null。

TreeMap類別

```
package ch9_15;
import java.util.*;
public class ch9_15 {
    public static void main(String[] args) {
        // Creating an empty TreeMap
         TreeMap<Integer, String> tree_map = new TreeMap<Integer, String>();
        // Mapping string values to int keys
         tree_map.put(10, "Geeks");
         tree_map.put(15, "4");
        // Displaying the TreeMap
         System.out.println("Initial Mappings are: " + tree_map);
        // Inserting existing key along with new value
         String returned_value = (String) tree_map.put(20, "All");
        // Verifying the returned value
         System.out.println("Returned value is: " + returned_value);
        // Displayin the new map
         System.out.println("New map is: " + tree_map);
    }
}
```

結果:

```
Initial Mappings are: {10=Geeks, 15=4}
Returned value is: null
New map is: {10=Geeks, 15=4, 20=All}
```

使用Iterator反覆運算器

- 反覆運算器（iterator），使用者可在容器物件（container，例如連結串列或陣列）
- 遍訪的物件，可幫助程式設計人員無需關心容器物件的記憶體分配的細節。

使用Iterator反覆運算器

- Java提供的ListIterator介面擴展了Iterator介面的功能，定義了迭代器Iterator的訪問方法

Iterator介面只提供了三個方法：hasNext()方法，next()方法和remove()方法。

- hasNext() 判斷是否遍歷到的最後一個元素；
- hasPrevious() 判斷是否還有上一個元素，有則返回true。
- next() 返回反覆運算器指向的下一個元素；
- remove() 刪除反覆運算器當前返回的元素。

使用Iterator反覆運算器

```
package ch9_16;
import java.util.*;
public class ch9_16 {
    public static void main(String[] args) {
        // 集合物件ArrayList宣告
         ArrayList<String> alist = new ArrayList<String>(4);
         alist.add("奇異冒險");  // 新增元素
         alist.add("冰雪奇緣");
         alist.add("魔法滿屋");
         System.out.println("ArrayList元素: " + alist);
        // 使用ListIterator介面顯示List元素
         System.out.print("List元素(ListIterator):");
         ListIterator<String> iterator =alist.listIterator(0);
         while (iterator.hasNext() )
             System.out.print(" " + iterator.next());
             System.out.println();
        // 使用ListIterator介面反向顯示List元素
             System.out.print("反向顯示元素(ListIterator):");
             ListIterator<String> iterator1 =alist.listIterator(alist.size());
         while (iterator1.hasPrevious() )
             System.out.print(" "+iterator1.previous());
             System.out.println();
    }
}
```

使用Iterator反覆運算器

結果:

```
ArrayList元素: [奇異冒險, 冰雪奇緣, 魔法滿屋]
List元素(ListIterator): 奇異冒險 冰雪奇緣 魔法滿屋
反向顯示元素(ListIterator): 魔法滿屋 冰雪奇緣 奇異冒險
```

使用Collections

- Collections類別是Java為方便進行Collection收集的操作和處理所提供的類別。
- Collections類別與Collection介面的區別：
- Collection介面是收集類別的根介面，提供對這類收集中的元素進行基本操作的通用方法。
- Collections類別是具體的實現類別，為實現Collection介面的收集提供了一系列靜態方法，完成元素的處理，如排序、查尋、特定變換等操作。

使用Collections

Collections類常用方法

返回類型	方法名	方法功能
static int	binarySearch(List list, T key)	使用二分搜索法搜索列表，返回key所在的位置。
static void	copy(List dest, List src)	將所有元素從清單src複製到列表dest。
static void	fill(List list, T obj)	使用元素obj替換列表list中的所有元素。
static T	max(Collection coll)	按自然順序，返回集合coll的最大元素。
static T	max(Collection coll, Comparator comp)	按指定比較器comp產生的順序，返回集合coll的最大元素。

使用Collections

返回類型	方法名	方法功能
static T	min(Collection coll)	按元素的自然順序 返回集合coll的最小元素。
static boolean	replaceAll(List list, T oldVal, T newVal)	使用元素newVal替換列表中出現的所有oldVal元素。
static void	reverse(List list)	反轉列表list中元素的順序。
static void	sort(List list)	按元素的自然順序對清單list按昇冪進行排序。
static void	sort(List list, Comparator c)	按指定比較器產生的順序對清單list進行排序。
static void	swap(List list, int i, int j)	將列表list中第i位置和第j位置的元素進行互換。

使用Collections

```
package ch9_17;
import java.util.*;
public class ch9_17 {
    public static void main(String[] args) {
            // 創建一個ArrayList
             ArrayList<String> list1 = new ArrayList<String>();
             list1.add("55");
             list1.add("04");
             list1.add("13");
             System.out.println("list1排序前:" +list1); //顯示結果
             Collections.sort(list1); //升序排列
             System.out.println("list1排序後" + " Collection.sort():" +list1);

   }
}
```

結果:

```
list1排序前:[55, 04, 13]
list1排序後 Collection.sort():[04, 13, 55]
```