包裝類別與工具類別(1)

TYIC桃高資訊社

包裝類別

雖然在 Java 中幾乎所有東西都是物件，但基本資料型別卻不是
這導致基本資料型別無法像物件一樣呼叫方法
所以出現了包裝類別(wrapper class)來解決這個問題
8 種基本資料型別對應了 8 種包裝類別，分別為：
Byte、Short、Character、Integer、
Long、Float、Double、Boolean
這些包裝類別皆位於 java.lang 套件內，所以可以直接使用

包裝類別

欲創建包裝類別，須呼叫包裝類別的公開靜態方法 "valueOf"
必有一個多載具有唯一參數，且為對應的基本資料型別
有些包裝類別有多載該方法，可能的參數有字串等
這個動作稱為裝箱(boxing)
而呼叫包裝類別的公開動態方法 "xxxValue"(xxx為基本資料型別)
將包裝類別變為基本資料型別
就被稱為拆箱(unboxing)

自動拆箱

包裝類別可以像基本資料型別一樣進行各式運算

```java
public class Main1 {
    public static void main(String[] args) {
        System.out.println(100 + Integer.valueOf(200));
        System.out.println(Integer.valueOf(100) / 200);
        System.out.println(Integer.valueOf(100) % Integer.valueOf(200));
    }
}
```

自動拆箱

這是因為編譯器會在包裝類別運算前呼叫

"xxxValue"(xxx為基本資料型別) 方法，稱為自動拆箱

```java
public class Main1 {
    public static void main(String[] args) {
        System.out.println(100 + Integer.valueOf(200).intValue());
        System.out.println(Integer.valueOf(100).intValue() / 200);
        System.out.println(Integer.valueOf(100).intValue() % Integer.valueOf(200).intValue());
    }
}
```

自動裝箱

將包裝類別賦值給基本資料型別的變數時也會自動拆箱

而將基本資料型別賦值給包裝類別的變數時則會自動裝箱

```java
public class Main2 {
    public static void main(String[] args) {
        final Integer TWO_HUNDRED = 100;
        System.out.println(add(100, TWO_HUNDRED));
    }

    public static Integer add(Integer a, Integer b) {
        return a + b;
    }
}
```

自動裝箱　　自動拆箱

```java
public class Main2 {
    public static void main(String[] args) {
        final Integer TWO_HUNDRED = Integer.valueOf(100);
        System.out.println(add(100, TWO_HUNDRED.intValue()));
    }

    public static Integer add(Integer a, int b) {
        return Integer.valueOf(a + b);
    }
}
```

陣列

考慮儲存 2 個學生的資料，可能可以宣告兩個變數來儲存

但考慮儲存 100 個學生的資料，宣告 100 個變數顯然不太現實

此時便可以使用陣列(array)來儲存多個相同型別的資料

陣列也是個物件，並且直接繼承 Object，但沒有覆寫任何方法：

```java
new 元素型別[]{元素1, 元素2, ..., 元素n} // 第一種，指定陣列內容
new 元素型別[陣列長度] // 第二種，無指定陣列內容
```

陣列裡每個儲存的值稱為元素(element)，陣列型別為 "元素型別[]"

第一種創建方式指定了陣列的內容和長度(length)

大括號內填入不定數量的元素，以逗號分隔，元素的數量即為陣列長度

第二種創建方式只指定了陣列長度，並沒有指定內容

兩種皆只能作為表達式

陣列

在賦值給陣列變數時，如果使用第一種(指定值)方法創建陣列

可以省略前方的 "new 元素型別[]"

```java
元素型別[] 變數名稱 = new 元素型別[]{元素1, 元素2, ..., 元素n};
元素型別[] 變數名稱 = {元素1, 元素2, ..., 元素n};
```

陣列沒有方法，但有一個不可變欄位 "length"，儲存陣列長度

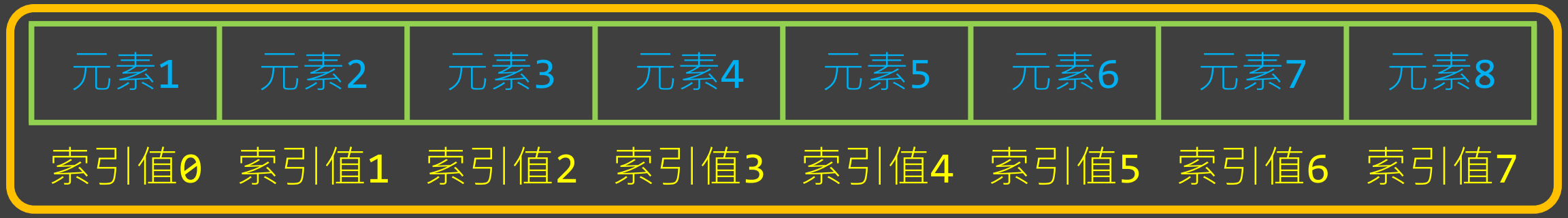
若要存取陣列的某個元素，需要透過下標運算子[] 來存取

下標運算可為陳述式或表達陳述式

```java
陣列[索引值]
```

可將下標運算整體視為一個變數，像變數一樣操作

索引值



索引值(index)是用來表示陣列的某個元素的位置

從 0 開始編號，到陣列長度 - 1，所以第 1 個元素索引值為 0

第 n 個元素索引值為 n - 1

```java
public class Main1 {
    public static void main(String[] args) {
        int[] arr = {6, 4, 5, 7, 2, 9};
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

```output
6 4 5 7 2 9
```

```java
public class Main2 {
    public static void main(String[] args) {
        int[] arr = {6, 4, 5, 7, 2, 9};
        arr[5]++;
        arr[4] = 0;
        for (int i = 0; i <= arr.length - 1; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

```output
6 4 5 7 0 10
```

若存取的索引值超過最大索引值，則會出現錯誤

增強 for

每次迭代(iteration)陣列都寫這麼長一個 for 迴圈，實在不便

所以可以使用「增強 for 迴圈」來避免寫這麼長的 for 迴圈

```java
for (陣列元素型別 變數 : 陣列) {
    陳述式...
}
```

其中變數會在每次循環依序變為陣列中的元素

```java
public class Main3 {
    public static void main(String[] args) {
        int[] arr = {6, 4, 5, 7, 2, 9};
        for (int e : arr) {
            System.out.print(e + " ");
        }
    }
}
```

```output
6 4 5 7 2 9
```

不定長度引數

不定長度引數(variable-length argument)

是指傳入的引數數量不限制，可以有很多個，使用不定長度參數接收

而一個方法只能有一個不定長度參數，且必須是最後一個參數

不定長度參數是一個陣列，內容為不定長度引數

不定長度參數也可以接收陣列，但互換則不行

```java
修飾子 返回值型別 方法名稱(參數型別 參數名稱, ..., 不定長度參數型別... 不定長度參數名稱) {
    陳述式...
}
```

下方為 java.io.PrintStream 的 printf 方法定義

其使用到不定長度參數來接收不定數量的物件

```java
public PrintStream printf(String format, Object ... args) {
    return format(format, args);
}
```

不定長度引數

```
153 289 51 459
17
```
console

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main5 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        int d = scanner.nextInt();
        System.out.println(gcd(a, b, c, d));
    }

    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    static int gcd(int... nums) {
        if (nums.length == 1) return nums[0];
        return gcd(nums[0],
                gcd(Arrays.copyOfRange(nums,
                        1, nums.length)));
    }
}
```
java

```
194 2716 582 1746 9506 388^D
0
```
console

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int index = 0;
        int[] arr = new int[200];
        while (index < 200 && scanner.hasNextInt()) {
            arr[index++] = scanner.nextInt();
        }
        System.out.println(gcd(arr));
    }

    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    static int gcd(int... nums) {
        if (nums.length == 1) return nums[0];
        return gcd(nums[0],
                gcd(Arrays.copyOfRange(nums,
                        1, nums.length)));
    }
}
```
java

陣列工具類別

陣列的工具類別是 java.util.Arrays

當中定義了許多關於陣列的公開靜態方法，如：

arrType copyOf(srcArray, newArrLength)、void sort(array)

void equals(arr1, arr2)、String toString(array)

void fill(array, value)、int binarySearch(array, value)

更多方法可以在 Java API 中查找

```java
import java.util.Arrays;

public class Main1 {
    public static void main(String[] args) {
        int[] arr1 = {1, 4, 7, 2, 5, 8, 3, 6, 9};
        int[] arr2 = Arrays.copyOf(arr1, arr1.length);
        System.out.println(Arrays.equals(arr1, arr2));
        Arrays.sort(arr1);
        System.out.println(Arrays.binarySearch(arr1, 2));
        System.out.println(Arrays.toString(arr1));
        int[] arr3 = new int[4];
        Arrays.fill(arr3, 6);
        System.out.println(Arrays.toString(arr3));
    }
}
```

```output
true
4
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[6, 6, 6, 6]
```

特別注意，呼叫 binarySearch 前一定要先將陣列排序(sort)

這與其查找原理有關

Java 21 API

k 維陣列

剛剛所介紹的其實叫做一維陣列(1D array)

由 1 個索引值確定元素

而二維陣列(2D array)由 2 個索引值確定元素

k 維陣列由 k 個索引值確定元素

在 Java 中，二維陣列就是元素型別為一維陣列的一維陣列

k 維陣列就是元素型別為(k - 1)維陣列的一維陣列

```java
public class Main4 {
    public static void main(String[] args) {
        int[][] arr = {{2, 1, 4}, {7, 4}, {8, 3, 6, 4}, {7}};
        for (int[] subArr : arr) {
            for (int e : subArr) {
                System.out.print(e + " ");
            }
            System.out.println();
        }
    }
}
```

```output
2 1 4
7 4
8 3 6 4
7
```

字串

在 Java 中，字串就是 java.lang.String 類別的實例

其有許多動態方法，如：int length()、String strip()

```
String[] split(String regex)
String substring(int beginIndex, int endIndex)
boolean startsWith(String prefix)
boolean endsWith(String suffix)
boolean contains(String s)、char charAt(int index)
String replace(String target, String replacement)
```

不管是哪個程式語言，字串通常都有這些方法

字串

```java
import java.util.Scanner;

// 找整數中連續 n 個數字和的最大值
public class Main3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int count = scanner.nextInt();
        String maxString = "";
        int max = -1;
        String string = scanner.next();
        for (int i = 0; i <= string.length() - count; i++) {
            int sum = 0;
            for (int j = 0; j < count; j++) {
                sum += Character.getNumericValue(string.charAt(i + j));
            }
            if (sum > max) {
                max = sum;
                maxString = string.substring(i, i + count);
            }
        }
        System.out.println(maxString + "," + max);
    }
}
```

java

```
5
2147483647
74836,28
```

console

列舉

列舉(enumerate)，顧名思義，就是把東西列出來

在 Java 中，列舉就是一個特殊的類別：

1. 列舉類別為 final 類別，無法被繼承
2. 其中的常數就是公開靜態不可變欄位，為該列舉類別的實例
3. 列舉類別的建構子為 private，外界不可實例化

```
修飾子 enum 列舉類別名稱 {常數1, 常數2, ..., 常數n}
```

```
修飾子 enum 列舉類別名稱 {
    常數1(引數...), 常數2(引數...), ..., 常數n(引數...);

    欄位...
    方法...
    建構子...
    類別...
}
```

列舉

列舉類別的公開靜態方法 Role values()

可以返回該列舉類別的常數組成的陣列

```java
public class Main {
    public static void main(String[] args) {
        for (Role role : Role.values()) {
            new Person(role).printInfo();
        }
    }
}

enum Role {
    WORKER("上班族"), BABY("嬰兒"), STUDENT("學生");

    final String description;

    Role(String description) {
        this.description = description;
    }
}
```

```java
class Person {
    Role role;

    Person(Role role) {
        this.role = role;
    }

    void printInfo() {
        System.out.println(role.description +
                "：" + switch (role) {
            case WORKER -> "上班";
            case BABY -> "哭";
            case STUDENT -> "上課";
        });
    }
}
```